ZX SPECTRUM

Программирование на

АССЕМБЛЕРЕ

RET , MOV , ADD , SUB
& ...

St. Petersburg, 1992

ОГЛАВЛЕНИЕ

Как ориентироваться в машинном языке 2
Начало 2
Основные понятия машинного языка 5
Как считает ЭВМ 9
Как представляется информация 13
Заглянем в ЦП 17
Все это хорошо, но как же мне выполнить программу на машинном языке ? 23
Как ЦП применяет свои конечности (регистры). 26
Откладывание чисел на одной руке 30
Флаги и их применение 35
Увеличение и уменьшение чисел на одной и двух руках 39
Арифметические операции для одной руки 41
Логические операторы 45
Работа с числами для двух рук 49
Обработка чисел на двух руках 51
Работа со стеком 55
Арифметические действия для двух рук 58
Циклы и переходы 60
Применение подпрограмм в ваших программах на машинном языке . . . 65
Операции над блоками 67

Более редко используемые команды Z80 70
Обмен регистрами 70
Установка и сброс битов 71
Сдвиги и циклические сдвиги 72
Ввод и вывод 75
Представление двоично-кодированных десятичных чисел 77
Прерывания 78
Команда рестарта (RST) 79

Написание программ для «SPECTRUM» 79
Планирование вашей программы на машинном языке 79
Средства «ZX SPECTRUM» 83

Введение в мониторные программы на машинном языке 88
Мониторная программа EZ CODE 88
Мониторная программа загрузки текста программы на машинном языке в шестнадцатеричном формате HEXLOAD 97
Приложение A 101
Приложение B 102
Приложение C 103
Приложение D 104
Приложение E 105
Приложение F 105
Приложение G 106
Приложение H 109
Приложение I 116

Как ориентироваться в машинном языке

Начало

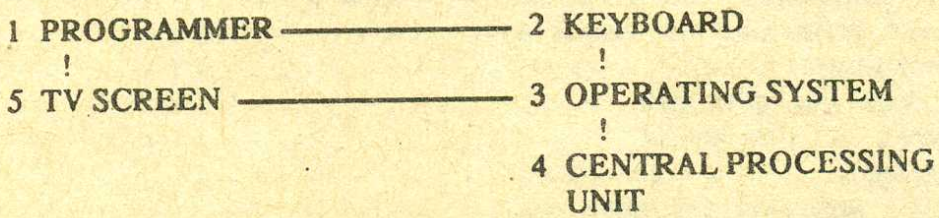
Эта книга создана в качестве введения в программирование на машинном языке и языке ассемблера для ЭВМ «SINCLAIR ZX SPECTRUM».

Может оказаться, что до чтения этой книги у вас не было ясного представления о программировании на машинном языке.

Вы можете даже не знать, что такое машинный язык. Вам может быть даже неизвестно, что есть разница между машинным языком и языком ассемблера, и чем они оба отличаются от программирования на языке BASIC.

Не беспокойтесь и не пугайтесь профессионального жаргона мы все постепенно объясним.

Прежде всего, давайте посмотрим, как работает ЭВМ:



1 — программист; 2 — клавиатура; 3 — операционная система; 4 — центральный процессор; 5 — телевизионный экран.

Цель этой диаграммы в том, чтобы показать, что между программистом и мозгом «SPECTRUM», центральным процессором, имеется барьер. В процессе обычной обработки программист не может задать центральному процессору, обычно называемому ЦП, что тот должен делать.

В ЭВМ «синклер» выбран ЦП типа Z80A, представляющий собой более быстродействующий вариант популярного чипа Z80. 4 чипа — Z80, 6502, 6809 и 8088, стали широко распространенными в качестве ЦП для микро-ЭВМ. Z80 — самый популярный из них.

Я уверен, вас не удивит, что Z80 не понимает слова на языке BASIC. На самом деле ни один чип не был разработан таким образом, чтобы мы могли непосредственно связываться с мозгом ЭВМ.

Если достаточно глубоко вдуматься, можно понять, что было бы очень трудно, если вообще возможно, в любом случае задать для чипа ЭВМ команду, имеющую смысл для человека. Снимите крышку вашего «синклера» (если решитесь!) и посмотрите на ближайший к громкоговорителю чип — это ЦП Z80A. Очевидно, этот чип вашей ЭВМ может только реагировать на электрические сигналы, передаваемые ему остальными частями схемы!

Что такое машинный язык?

Чип Z80 сконструирован таким образом, что он может принимать сигналы одновременно с восьми подсоединенных к нему контактов.

Разработчики чипа Z80 сконструировали его таким образом, что разные комбинации поступающих на него с этих восьми контактов сигналов будут «командовать» Z80 выполнять разные функции.

Помня о том, что на самом деле есть только электрические сигналы, давайте примем систему условных обозначений этих сигналов — например, ставя «1», если на одном из контактов есть сигнал, и «0», если сигнала нет.

Типичная команда могла бы тогда выглядеть так:

```
00111100
```

Это существенно отличается от

«LET A = A + 1»,

например, не так ли!

Тем не менее, именно это и есть машинный язык. Название говорит само за себя! Это — язык для машин. Каждый производитель различных чипов сконструировал для своей продукции свой «особый» язык!

Здесь вы можете задать себе вопрос, если это и есть машинный язык, то в чем проблема? Почему не воспользоваться проделанной кем-то другим работой, что позволит мне программировать на ЭВМ на языке, который я легко понимаю, таком как BASIC или «кобол»?

В этом есть определенный смысл из-за следующих преимуществ машинных языков:

Более быстрое выполнение программы;

Более эффективное использование памяти;

Более короткие программы (в памяти);

Независимость от операционной системы.

Все перечисленные выше преимущества — непосредственное следствие программирования на языке, который ЦП воспринимает без предварительной трансляции. Когда вы программируете на «Бейсике», то на самом деле ЭВМ выполняет программу операционной системы, написанную на машинном языке. Эта программа выглядит примерно так:

NEXT — взять следующую команду,
Перевести ее в последовательность команд машинного языка,
Выполнить каждую команду,
Записать результат, если это нужно,
Снова перейти на NEXT

Если вас интересует, где ЭВМ берет эту программу, операционную систему, то она хранится в ПЗУ. Иными словами, она встроена в «SPECTRUM». (ПЗУ — сокращенное название постоянного запоминающего устройства машинной памяти, ячейки которой вы не можете менять, а можете только читать (PEED/PEEKED).

Программа на языке BASIC может выполняться почти в 60 раз медленнее, чем программа, написанная непосредственно на машинном языке!

Причина этого — в том, что трансляция отнимает время, а также получающиеся в результате команды машинного языка менее эффективны. Аналогичным образом, обычно более быстрым оказывается личный автомобиль, а не общественный транспорт; вы можете сократить путь по известным вам дорогам, а не следовать маршрутом общественного транспорта, которому приходится удовлетворять общественные потребности.

Тем не менее нам приходится первыми признать, что у программирования на машинном языке есть свои недостатки.

У машинного языка есть следующие основные недостатки:

Программы трудно читать и отлаживать;

Их невозможно перенести на другие ЭВМ;

Программы оказываются длиннее (в командах);

Трудно выполнять арифметические вычисления.

Это означает, что вам следует очень продуманно принимать решение о том, какой метод программирования следует использовать для каждой конкретной задачи.

Очень большую программу из области финансов следует писать на языке, разработанном для обработки чисел и таком, что программы в случае необходимости можно легко модифицировать.

С другой стороны нет ничего хуже, чем игра для аркады (аркада — помещение для игры на ЭВМ. Примеч. пер.) написанная на языке «Бейсике», — когда вы сядете играть, она окажется слишком медленной.

Ваш выбор языка программирования будет определяться вашими собственными потребностями. Объемом памяти вашей ЭВМ, требующимся временем реакции, отпущенным на разработку временем и т.д.

Итак, подведя итог, машинный язык — последовательность команд, понятных для ЦП и поддающихся представлению в виде чисел.

Что такое язык ассемблера?

Совершенно очевидно, что если бы машинный язык можно было представлять только с помощью чисел, очень немногие люди были бы способны писать программы на машинном языке.

В конце концов, кто смог бы понять смысл программы, имеющий следующий вид:

```
00100001
00000000
01000000
```

и т.д.

К счастью, мы можем придумать ряд имен для каждого из этих чисел. Язык ассемблера как раз и является таким представлением машинного языка, так что люди могут читать его в более понятной форме, чем

```
01110111
```

Между языком ассемблера и машинным языком есть только одно различие: язык ассемблера на один уровень выше, чем машинный язык. Его легче читать человеку, чем машинный язык, но с другой стороны, ЭВМ не может читать язык ассемблера.

Он не является адаптацией машинного языка, подобно «Бейсику». Для каждой команды языка ассемблера имеется идентичная (по функции) команда машинного языка и наоборот. Иными словами, между ними имеется взаимно однозначное соответствие. Поэтому можно сказать, что язык ассемблера эквивалентен машинному языку.

Язык ассемблера использует мнемонику (или сокращения) для повышения удобочитаемости. Например, на данной стадии обучения команда

```
INC HL
```

может почти ничего не означать для вас, но, по крайней мере, вы можете ее прочитать. Если бы вы сказали, что «INC» — стандартное сокращение (или мнемоническое обозначение) INCREASE (увеличить), а HL — «переменная», то просто посмотрев на эту команду вы можете получить представление о том, что происходит. Та же самая команда на машинном языке имеет вид

```
00100011
```

Теперь, очевидно, вы можете также «прочитать» эту команду в том смысле, в котором вы можете прочитать число. Это не будет много для вас означать, если только у вас нет справочной таблицы или ваш мозг не действует почти как ЭВМ.

Язык ассемблера может быть преобразован непосредственно в машинную программу с помощью программы или вами самими. Такая программа называется «Ассемблером». Вы можете рассматривать ее как программу, выполняющую довольно утомительную задачу трансляции вашей написанной на языке ассемблера программы в последовательность команд машинного языка, понятных «SPECTRUM» и мы считаем, что «ассемблер» для «ZX SPECTRUM» уже имеется.

Тем не менее, такие ассемблеры обычно требуют 6К памяти и имеют ограниченное применение на ЭВМ с объемом памяти 16К. Дисплей «SPECTRUM» отнимает 7К памяти, и

после загрузки ассемблера у вас может остаться всего 4K памяти для программы на языке ассемблера (это означает примерно 1/2K программы на машинном языке).

Альтернативный способ работы — вместо применения программы ассемблера самому транслировать мнемонику языка ассемблера в машинный язык вручную, применяя приведенные в этой книге таблицы.

Это трудно, поначалу кажется безнадежно, неудобно, но это — прекрасная практика и дает вам глубокое понимание того, как работает ЦП «SPECTRUM».

Мы на самом деле рекомендуем, чтобы вы попытались написать таким способом короткую программу на машинном языке т.е. написать их на языке ассемблера и вручную транслировать на машинный язык — прежде чем покупать программу ассемблера.

Заключение

ЦП

Центральный процессор ЭВМ. Это — чип, выполняющий в ЭВМ вычислительные и управляющие функции.

Машинный язык.

Язык, воспринимаемый ЦП. Для ЦП «SPECTRUM» это — машинный язык Z80, содержащий приблизительно 200 команд.

Язык BASIC

Язык программирования, разработанный так, чтобы быть понятным человеку. Когда ЭВМ выполняет команду на языке BASIC, ей приходится транслировать эту команду в последовательность команд машинного языка. Поэтому программы на языке BASIC значительно медленнее работают, чем программы на машинном языке, но их легче писать.

Язык «ассемблера»

Представление команд машинного языка в сокращенной, понятной человеку записи. Например, HALT — ассемблерный эквивалент команды машинного языка 0 1 1 1 0 1 1 0.

Программа ассемблера

Программа, транслирующая команды языка ассемблера (удобочитаемые и понятные человеку) в команды машинного языка (воспринимаемые ЭВМ. Например, «SPECTRUM»).

Постоянное запоминающее устройство (ПЗУ)

Большая программа на машинном языке, обычно называемая программируемым оборудованием; программа, жестко встроенная в аппаратуру ЭВМ; она сохраняется даже при отключении питания. Для «SPECTRUM» ПЗУ запрограммировано в машинных кодах Z80 и написано специально для него. ПЗУ «SPECTRUM» занимает ячейки памяти с 0 по 16383. Вы можете только обращаться к содержимому этих ячеек, тогда как остальную память вы можете не только читать, но и изменять, как требуется.

Основные понятия машинного языка

Что такое ЦП ?

Если мы хотим обмениваться информацией с ЭВМ, то нам необходимо знать, какого типа команды она будет воспринимать, и на каком языке разговаривает мозг ЭВМ (ЦП).

Если мы не знаем, информацию какого типа воспринимает ЦП, то мы не сможем как следует объяснить ЭВМ, какие замечательные задачи она должна для нас решать — от соперничества в шахматной игре до бухгалтера, следящего за нашими счетами.

В ЦП нет ничего загадочного. Мне нравится представлять ЦП в виде одинокого человечка, сидящего внутри вашего «SPECTRUM»а, которого все время просят что-то делать. В особенности — вычислять.

Но у бедного человечка нет даже карандаша и бумаги для записи происходящего. Как же он делает это?

Конструкция ЦП

Сейчас мне, наверное, следует рассказать вам, как конструкторы Z80 представляют объекты, и как ЦП полагается обрабатывать их. ЦП сконструирован для выполнения очень простых заданий, но он способен выполнять их очень быстро.

Мы упомянули, что у ЦП нет даже карандаша и бумаги, и именно так он сконструирован. Любое число, которое он не может запомнить или найти, ЦП должен положить в какой-нибудь ящик на хранение.

Давайте рассмотрим один пример, скажем, вы хотите, чтобы ЦП определил время в Нью-Йорке, зная время в Лондоне.

Теперь, считая, что ЦП ничего не знает, вам прежде всего придется сказать ему, который час в Лондоне, скажем, 10 часов. ЦП негде запомнить эту информацию и неизвестно, что вы попросите его делать дальше, так что он откладывает эту информацию в ящик, скажем, ящик N1.

Затем вы должны сообщить ему разницу во времени между Нью-Йорком и Лондоном, скажем, на пять часов раньше, он откладывает ее в ящик N2.

Приходит время для вычислений. Он спешит к ящику N1, достает число, идет к ящику N2, выполняет вычисления и откладывает результат, скажем, в ящик N3.

10 - 5 = 5

Ответ, конечно, 5 часов.

Вся эта беготня между ящиками, сложение, вычитание и т.д. были бы очень утомительными, если бы ЦП нужно было все делать в уме, поэтому он поступает точно как вы и я — считает на пальцах рук и ног.

Руки и ноги ЦП называются регистрами.

Чип Z80 в вашем «SPECTRUMe» отличается тем, что у него масса рук и ног — но мы остановимся на этом позже.

Чтобы показать, как в точности ЦП вычисляет разницу во времени в приведенном выше упражнении, давайте назовем одну из рук ЦП «рука А». Как ЦП управляет содержимым ящика N1 и ящика N2 ?

Приводимая ниже последовательность довольно точно соответствует тому, что на самом деле делает ЦП при заданных выше командах:

Отложить значение из ящика N1 на пальцах руки А;

Вычесть содержимое ящика N2 из того, что у него уже отложено на пальцах;

Взять количество пальцев на руке А и поместить его в ящик N3.

Итак, если именно так происходит, отсюда вытекает, довольно неожиданные выводы:

1. ЦП не сможет обрабатывать числа, подобные 11, 53 — он может обрабатывать только целые числа.

2. ЦП в своих вычислениях будет ограничен числами, которые он откладывает на своих пальцах. Это, конечно, верно.

Есть, однако, утешение что у ЦП масса рук и ног, и он может на всех из них считать по отдельности, и что он может считать до 255 с помощью всего 8 пальцев на руке А.

В следующей главе мы подробнее рассмотрим, почему ЦП может считать больше, чем до 8 на каждой руке, тогда как мы с помощью двух рук — только до 10! пока достаточно сказать, что на каждой руке можно считать до 255, а на каждой ноге — более чем до 64 000!

Упражнение с разницей во времени до сих пор не представлено на чем-либо, похожем на язык, воспринимаемый ЦП — мы только описали процесс.

Чтобы вы заранее получили представление о захватывающих аспектах программирования на машинном языкс, давайте теперь применим мнемонику (сокращения) чтобы на каждом шаге давать команду ЦП:

Установка:

```
LD (BOX N1), 10 1:LOAD BOX 1 WITH 10
LD (BOX N2), 5 2:LOAD BOX 1 WITH 5
```

1 — загрузить 10 в ящик N1: 2 — загрузить 5 в ящик N2

Вычисления:

```
LD A, (BOX N1) 1: LOAD A WITH BOX 1 CONTENTS
SUB A, (BOX N2) 2: SUBTRACT CONTENTS OF BOX 2
```

1 — загрузить содержимое ящика N1 в А: 2 — вычесть содержимое ящика N2

Запоминание результата:

```
LD (BOX N3), A 1: LOAD BOX 3 WITH A VALUE
```

1 — загрузить значение А в ящик N3

Эти команды сначала могут показаться излишне краткими, но, в конце концов, мнемоника есть мнемоника.

«LD» — сокращенное LOAD (загрузить), так что «LD A,1» например, означает загрузить 1 в А, т.е. отложить «единицу» на пальцах руки А.

В этих мнемонических обозначениях используется также довольно разумный образ, основанный на применении скобок: скобки используются, чтобы показать, что мы хотим использовать содержимое того, что стоит в скобках.

Это должно быть нетрудно запомнить с помощью наглядного представления, поскольку скобки выглядят так, что напоминают контейнер.

Таким образом проходя приведенные выше мнемонические обозначения, мы загружаем в ящик N1 и N2 10 и 5, и т.д... и получаем конечный результат, равный 5, в ящике N3.

Всё это достаточно просто для понимания, и я уверен, вы понимаете, что пока вы выполняете вычисления, числа на руке А используются для представления времени в Нью-Йорке. Минутой позже они могут использоваться для представления количества служащих в фирме, а в некоторый другой момент — количества имеющихся у вас денег.

Если вы привыкли к понятию переменной, программируя на языке «Бейсик», то при программировании на машинном языке вы должны отказаться от него.

Пальцы на руке «А» — это не переменная в том смысле, в котором она используется в программе на «Бейсике». Это — просто то, с помощью чего ЦП считает.

Одно из существенных различий между программированием на машинном языке и на языке BASIC — это отсутствие переменных.

Вы можете считать, что ящики, используемые нами для запоминания информации, аналогичны переменным BASIC, если мы каждому дадим имя.

Да, вы абсолютно правы, но это все-таки не переменные. Они могут быть очень полезны и выполнять те же функции хранения, что и переменные, но имейте в виду, что эти ящики — не более, чем ячейки памяти, отведенные для конкретных целей.

Несколько иначе ЦП обрабатывает отрицательные числа, но этот вопрос мы рассмотрим позднее.

Что, если ЦП не хватит рук ?

Нужно сказать, что ЦП, встреть вы его на улице, показался бы вам, наверное, очень странным субъектом.

На каждой из его рук по восемь пальцев, а самих рук восемь! У него только две ноги, но на каждой по 16 пальцев, и он ими очень живо перебирает!

Поэтому он хорошо подходит для большого количества вычислений, для которых он предназначен, откладывая все числа на пальцах рук и ног.

Тем не менее, возможно, что в некоторых случаях у ЦП окажется недостаточно рук для выполнения необходимых вычислений или по некоторым причинам программист захочет остановить ЦП посреди вычислений, чтобы выполнить что-то другое.

ЦП Z80 выходит из положения, применяя стек, то есть одну из тех высоких колючих вещей, которые некоторые держат на столе для хранения счетов, записок и т.п. Я уверен, вы видели такие стеки, на которые вы накалываете сначала один кусок бумаги, потом следующий и т.д. Это удобная система хранения, если вам нужен только верхний клочок бумаги, но очень неудобная, если нужен клочок из середины, так как придется перерыть все бумаги в стеке.

Получилось, что это — очень удобная система для ЦП, поскольку ему всегда нужна верхняя порция информации.

Каждый раз, когда прерывание заставляет ЦП прервать вычисления, он «вталкивает» всю информацию со своих рук в «стек», а когда прерывание закончено, «выталкивает» верхние порции информации и продолжает свою работу.

В терминологии ЭВМ мы называем иглу «стеком». Когда мы помещаем порцию информации в стек, мы «вталкиваем» ее, а извлекая, мы «выталкиваем».

Втолкнуть в стек и вытолкнуть можно информацию любого типа, например, в середине сложных вычислений ЦП может потребоваться запомнить всю информацию с его многочисленных рук и ног, и для этого потребуется много отдельных «вталкиваний».

По причинам, лучше известным разработчикам Z80, наш ЦП предпочитает держать стек кверху ногами. Это значит, что чем больше информации «вталкивается» в стек, тем дальше вниз растет стек.

Основное преимущество применения стека для хранения временных порций информации состоит в том, что ЦП не приходится запоминать, в каком ящике находится информация — он знает, что это — последняя порция информации «втолкнутая» в стек. Естественно, требуется некоторый порядок, если нужно «вталкивать и выталкивать» много порций информации.

Что может ЦП ?

Я думаю, на этом этапе стоит рассмотреть, команды какого типа разработчики посчитали полезным встроить в чип Z80.

Поскольку ЦП должен иметь возможность отслеживать все свои вычисления на пальцах рук и ног, есть только два типа чисел, с которыми может работать ЦП:

Числа для одной руки — т.е. числа, которые можно отложить на пальцах одной руки;

Числа для двух рук — т.е. числа, которые можно отложить на пальцах двух рук.

Вам может оказаться трудно в это поверить, но ЦП не может обрабатывать числа, превосходящие те, которые он может отложить на двух руках! Типы команд, которые может выполнять ЦП, также очень ограничены: Откладывание чисел на одной руке; Откладывание чисел на двух руках;

Сложение, вычитание, увеличение, уменьшение или сравнение чисел для одной руки; сложение, вычитание, увеличение и уменьшение чисел для двух рук; различные преобразования чисел для одной руки — например, изменение знака числа; указание ЦП перейти к другой части программы; попытка обмена числами для одной руки с внешним миром. Я уверен, вы согласитесь, что это — очень ограниченный набор команд, и все же применяя лишь такой ограниченный набор вы можете заставить ЦП играть в шахматы и считать свою зарплату!

Обратите внимание, что нет даже таких простых команд, как умножение! Если вам нужно перемножить два числа на машинном языке, вам придется для этого писать программу.

Выводы

Регистры

У ЦП имеются регистры, которые он может использовать для вычислений. Восемь из них можно считать руками ЦП, а два его ногами. На каждой «руке» по 8 пальцев, а на каждой ноге по 16.

Ячейки памяти

ЦП может передавать информацию с одной своей руки на другую, а также в память и из нее.

Программист может отвести конкретные ячейки памяти для представления конкретной информации.

Стек

ЦП может использовать стек для передачи информации, которую программист хочет временно запомнить. Информация передается в стек путем «вталкивания» в него, а извлекается путем «выталкивания».

Возможные команды

ЦП может выполнять только команды такого типа, как простейшие передачи информации и простые арифметические действия. Все программы должны быть составлены из последовательности этих простейших команд.

Как считает ЭВМ

Выше упоминали, что ЦП может считать до 255 с помощью всего лишь восьми пальцев. Как это выходит, если с помощью десяти пальцев нам удается считать только до 10?

Так получается, конечно, не потому, что ЭВМ умнее нас (это не так), а потому, что информация в ЦП лучше организована, у нас: почему загнутый указательный палец имеет то же самое значение, что и мизинец (=«1»)?

Кажется очевидным, что при желании вы таким способом могли бы представить два различных числа.

Это во многом похоже на понимание того, что число 0 0 1 отличается от числа 1 0 0. Простая истина состоит в том, что люди не очень эффективно используют пальцы для счета.

ЦП понимает, что отсутствие пальца несет некоторую информацию, и то, какой палец отогнут, — ценная порция информации.

С помощью всего двух пальцев можно разработать способ счета от 0 до 3 таким методом:

0 0 - 0	1. WE CAN INDICATE NOT HAVING A FINGER RAISED AS ''0''
0 1 - 1	AND HAVING A FINGER RAISED AS ''1''.
1 0 - 2	2. THIS DOES NOT MEAN 11 - 3
1 1 - 3	IT MEANS WE SHOSE TO LET THE REPRESENTATION 11 (OR TWO FINGERS) HAVE THE VALUE 3.

1 — мы можем отмечать загнутый палец цифрой «0», а поднятый палец — цифрой «1»; 2 — это не означает, что 1 1 = 3, просто мы решили, что комбинация 1 1 (или два пальца) будет иметь значение 3

Так же просто мы могли бы выбрать и другой способ обозначения.

Между этим представлением и двоичной системой счисления имеется непосредственная связь. Пальцы ЦП — это ячейки памяти и можно сделать так, чтобы они обозначали «включено» и «выключено» (или «0» и «1», как требует система обозначений).

Если бы в приведенном нами выше примере мы добавили третий палец, то мы могли бы представить все числа от 0 до 7. Три пальца на все числа от 0 до 7!

С помощью четырех пальцев можно представить все числа от 0 до 15! Если вы этому не верите, то будет хорошим упражнением выписать все возможные комбинации из 4 пальцев.

Чтобы упростить систему обозначений таких чисел и избежать путаницы при попытке записать число одиннадцать так, чтобы оно отличалось от установки двух битов, было принято общее соглашение.

Числа от 10 до 15 обозначаются буквами A — F

Десятичное

10 = A
11 = B
12 = C
13 = D
14 = E
15 = F

Это означает, что мы следующим образом записываем десятичные числа от 0 до 15:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Не правда ли, просто?

Такой способ обозначения чисел называется шестнадцатеричным форматом.

Чтобы избежать путаницы некоторые пишут «H» после шестнадцатеричного числа (например, 10H). Это «H» не имеет значения, а служит просто для напоминания пользователю о шестнадцатеричной системе счисления.

При программировании на машинном языке удобно иметь дело с числами в шестнадцатеричном формате.

Это — только система обозначений и при желании вы могли бы записать все ваши команды в обычном десятичном формате. Нам удобно использовать шестнадцатеричный формат по следующим причинам:

1. Из этой формы числа легко перевести в двоичный формат, который говорит нам, что означает каждый бит (или палец).

2. Он дает нам возможность легко определить, будет ли число для одной руки или для двух рук — т.е. восьмиразрядное или шестнадцатиразрядное.

3. Он дает стандартное представление всех чисел в виде последовательности двухразрядных чисел (мы остановимся на этом подробнее).

4. Это общепринятая система обозначений и знакомство с шестнадцатеричной системой позволит вам с большей легкостью читать другие книги и руководства.

5. Поскольку ЦП сконструирован для обработки информации, представленной двоичными числами, которые людям для чтения неудобны, требуется более удобочитаемое представление.

Но это — только система обозначений, а не священное правило.

Шестнадцатеричная система, как отмечалось выше, позволяет представлять числа от 0 до 15 с помощью всего 4 битов. Любая 8-разрядная ячейка памяти или 8-разрядный регистр может поэтому быть описана как два набора по 4 бита (это — то же самое, что сказать, что любую комбинацию из 10 пальцев можно представить в виде дух рук).

Нас интересуют 8-разрядные ячейки памяти и 8-разрядные регистры потому, что такова структура «ZX SPECTRUM».

Все ячейки памяти и все одинарные регистры имеют по 8 разрядов. Понять это нетрудно — это все равно, что сказать, что у всех людей по 5 пальцев на руках.

Продвигаясь постепенно, давайте познакомимся сначала с комбинациями из 4 пальцев:

```
1 1 1 1   2**3  2**2  2**1  2**0
            8     4     2     1
```

1. DECIMAL 15
2. F (IN HEXADECIMAL NOTATION)

1 — десятичное 15; 2 — в шестнадцатеричной системе обозначений

Имеющие склонность к математике могут заметить, что представляемое каждым пальцем число каждый раз увеличивается в 2 раза, если двигаться справа налево. Если перенумеровать пальцы:

3 2 1 0 то значение каждого пальца равно «2 в степени N», где «N» — номер пальца. Будем называть руку с 4 пальцами «ручонкой» (точно так же, как небольшую собачку — собачонкой).

Упражнение

Какие десятичные и шестнадцатеричные значения представляют следующие комбинации битов (или пальцев)?

Двоичное	Десятичное	Шестнадцатеричное
0 0 1 0		
0 1 1 0		
1 0 0 1		
1 0 1 0		
1 1 0 0		

Вам важно познакомиться с шестнадцатеричной системой обозначений, и если это понятие вызвало у вас трудности, прочтите еще раз несколько последних страниц, прежде чем идти дальше.

Рассмотрим, что получится, если нам нужно число, превышающее 15, скажем, 16? Мы используем следующий палец слева так:

```
7 6 5 4 3 2 1 0
```

=16 десятичное = 10H (шестнадцатеричное) — причина, по которой мы пишем число в виде 10H, состоит в том, что мы делим руку на две «4-разрядные ручонки». Поэтому мы легко можем обозначить каждую ручонку одной из шестнадцатеричных цифр от 0 до 15 (0—9 и A—F).

Таким способом произвольную 8-разрядную руку можно записать в виде ровно двух шестнадцатеричных ручонок:

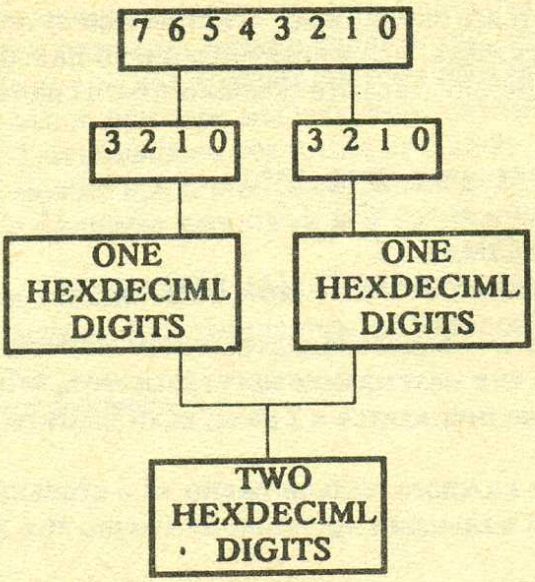
1 — одна шестнадцатеричная цифра; 2 — две шестнадцатеричных цифры.

Значение «ручонки» слева в 16 раз превышает значение «ручонки» справа. Это во многом напоминает десятичную систему счисления, где цифра в разряде «десятков» значит в десять раз больше, чем цифра в разряде «единиц».

Мы машинально преобразуем числа в десятичной системе, такие как 15, к виду: 15 =(1*10) + 5

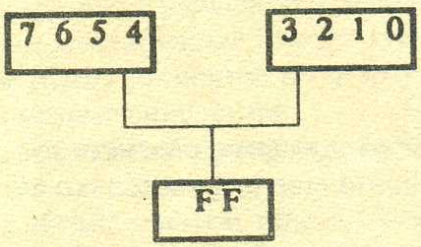
Это делается настолько машинально, что мы даже не думаем об этом.

Совершенно то же самое происходит в шестнадцатеричной системе счисления. Чтобы сделать обратное преобразование из шестнадцатеричной системы в десятичную, мы умно-



жаем шестнадцатеричное число в левой «ручонке» на 16. Используем приведенный выше пример: 10H = (1*16) + 0 = 16 десятичное

Именно таким способом мы получаем возможность считать до 255 с помощью всего 8 пальцев. Максимальное значение получается, когда все пальцы подняты:



FFH = (F*16) + F
= (15*16) + 15 (IN DECIMAL)
= 255 (DECIMAL) 1

1 — десятичное

Наименьшее число получается, когда все пальцы согнуты:

00H = 0 десятичное

Обратите внимание, что все числа, от наименьшего до наибольшего требуют двух и только двух цифр, чтобы определить число.

Возьмите сами произвольную комбинацию 8 двоичных разрядов и попытайтесь преобразовать ее сначала в шестнадцатеричную, а потом в десятичную систему счисления.

Сначала это может показаться странно и неудобно, но вы скоро к этому привыкните.

Точно так же, счет в шестнадцатеричной системе происходит тем же способом, что и в десятичной: десятичная: 26 27 28 29 30 и т.д. Шестнадцатеричная: 26 27 28 29 2a 2в 2c 2D 2e 2F 30 и т.д....

Значения чисел в приведенной выше десятичной и шестнадцатеричной последовательностях, конечно, различны. Обратите внимание, что после 29H у вас идет 2AH, а не 30H!

Следующая программа на языке BASIC позволит вам ввести в ваш «SPECTRUM» десятичное число и преобразовать его в шестнадцатеричное значение.

```
100 REM DECIMAL TO HEXADECIMAL CONVERSION
110 PRINT "PLEASE INPUT DECIMAL VALUE"
120 INPUT N : PRINT N
130 LET S$ - "": LET N2 = INT(N/16)
140 LET N1 - INT(N-N2*16)
150 LET S$ - CHR$ ((N1<=9)*(N1+48)+(N1>9)*(55+N1))+S$
160 IF N2 = 0 THEN PRINT : PRINT "HEXADECIMAL - 0" : S$ :"H"
      : FOR I = 1 TO 200: NEXT I: RUN
170 LET N = N2 : GO  TO 140
```

I — преобразование десятичных в шестнадцатеричные: 2 — введите пожалуйста, десятичное значение; S — знак заменен на знак — $

Попробуйте преобразовать в шестнадцатеричную систему следующие числа и с помощью программы на языке BASIC проверьте свой ответ.

1. 16484 адрес начала дисплейного файла «SPECTRUM»:
2. 22528 адрес начала файла атрибутов «SPECTRUM»:
3. 15360 адрес начала набора литер «SPECTRUM»:
4. 15616 адрес начала литер в коде ASCII «SPECTRUM».

Выводы

Десятичная система — это условные обозначения, позволяющие записывать числа группами по десять единиц. Эти группы представляются цифрами 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Шестнадцатеричная система счисления

Шестнадцатеричная система — это условные обозначения, позволяющие записывать группами по 16 единиц. Эти группы представляются цифрами 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Иногда в конце шестнадцатеричного числа добавляется литера «H» как напоминание о формате представления числа. Например, 1800H.

8-ми разрядные ячейки памяти

Конструкция «ZX SPECTRUM» такова, что в каждой ячейке памяти содержится по 8 битов («пальцев»). В каждой ячейке памяти может храниться число от 0 до 255 (десятичное). Его удобно представлять в шестнадцатеричном формате в виде двухразрядного числа.

Как представляется информация

В представлении информации человеком и ЭВМ имеется существенная разница. У человека информация в основном состоит из чисел и литер (алфавитно-цифровая информация), тогда как вся информация в ЭВМ хранится в виде групп битов.

Бит означает двоичный разряд (BINARI DIGJT. «0» или «1»); В микропроцессоре Z80A эти биты сгруппированы по 8. Группа из восьми битов называется «байтом».

Такой способ представления информации с помощью двоичных разрядов называется «двоичным форматом». Такова структура языка, на котором разговаривает Z80 и большинство ЦП микро-ЭВМ.

В основном имеется два типа информации, представляемой внутри «SPECTRUM». Первый — это программа, второй — данные, над которыми программа будет действовать и которые могут включать числа или алфавитно-цифровой текст. Мы так и будем ниже рассматривать эти три представления: программа, числа, алфавитно-цифровой текст.

Представление программы

Программа — последовательность команд ЦП выполнить конкретное задание, которое можно разбить на некоторое число «подразделений».

В Z80 все команды имеют внутреннее представление в виде одного или нескольких байтов. Команды, представленные одним байтом, называются «короткими командами». Более длинные команды представляются двумя или несколькими байтами.

Поскольку Z80 — восьмиразрядный микропроцессор, он может за один раз обрабатывать только один байт, и если ему требуется более одного, он ведет последовательный поиск байтов в памяти. Поэтому в общем случае однобайтовая команда будет выполняться быстрее, чем двух- или трехбайтовая. Таким образом, как правило выгодно писать программу на машинном языке, применяя, где можно, однобайтовые команды.

Вы можете посмотреть «короткие и длинные» команды в таблице кодов команд в приложении. Пусть вас не беспокоит, что они непонятны, позже мы рассмотрим каждую команду более глубоко.

Представление числовой информации

Представление целых чисел

Выше мы отмечали, что из-за способа конструирования Z80 мы не можем представлять такие числа, как 11, 53. ЦП может работать только с целыми числами. Кроме того, используя только 8 пальцев (т.е. 8-разрядные числа), мы могли бы представлять все числа из диапазона от 0 до 255. Например, десятичное 255 представляется как FFH, или в двоичном виде: 11111111. Ну, а как с отрицательными числами?

Представление целых чисел со знаком

Напомним, что байт — это рука с восемью пальцами, и число представляется с помощью отгибания различных пальцев.

Очевидно, чтобы представить целое число со знаком в двоичном формате, нам необходим некоторый способ представления положительных и отрицательных чисел. Давайте считать, что для представления отрицательного числа мы принимаем следующую систему обозначений (представление со знаком): число, представленное на руке ЦП, будет считаться отрицательным, если у ЦП отогнут большой палец. (В терминологии ЭВМ старший бит — бит 7 — установлен).

Итак, у нас остается всего семь пальцев (битов) для представления значения числа. Это значит, что наибольшим числом, которое можно представить, уже не будет 255. На самом деле половина чисел, которые могут содержаться на одной руке (в одном байте) будут отрицательными, а другая половина — положительными, (в зависимости от того, загнут большой палец или нет).

Полный диапазон чисел, представимых на одной руке, если допускаются отрицательные числа, буде тогда от -128 до +127. (обратите внимание, что весь диапазон чисел, которые можно представить, по-прежнему будет состоять из 256 чисел).

И вот здесь возникает трудность: когда число с поднятым большим пальцем будет положительным числом, а когда отрицательным?

Ответ будет: все зависит от вас; вы сами должны сделать выбор: числа могут лежать либо в диапазоне от 0 до 255, либо от -128 до +127. Но никак не в двух диапазонах сразу! Это вы, программист, должны решить, какую систему обозначений применять в конкретный момент.

Все команды будут действовать одинаково хорошо независимо от того, предпочтете ли вы, чтобы все числа в регистрах памяти были положительными или чтобы они были и положительными, и отрицательными.

Выбор представления отрицательных чисел

Мы уже решили, что подъем большого пальца будет обозначать отрицательное число, а его загибание — положительное. Достаточно ли этого?

Нет. Мы должны решить, какая из 127 возможных комбинаций остальных 7 пальцев будет обозначать -1, какая — -2 и так далее.

Нам необходимо представление отрицательных чисел такое, что при сложении числа с противоположным ему будет получатся ноль. В качестве упражнения, давайте подумаем, какое число при сложении с 1 дает ноль (это, очевидно, будет -1, и мы уже знаем, что большой палец — бит 7 — будет поднят):

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 ? ? ? ? ? . (COULD IT BE-)? 1 0 0 0 0 0 0 1
──────────────── 1 ────────────────
0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
```

1 — можно ли так?

Давайте попробуем 1 0 0 0 0 0 0 1, иными словами, то же самое, что и +1, но с поднятым большим пальцем. Чтобы проверить, будет ли это -1, попробуем сложить его с +1. Из вышеизложенного следует, что сумма 10000010, очевидно, не даст правильного ответа! Если бы это было нужное нам число, то ответ равнялся бы 00000000. Очевидно, нам нужно число, принимающее перенос из бита 0 и преобразующее его в последовательность нулей.

Вы можете попробовать сами проделать это, и вы увидите, что единственным числом, дающим правильный ответ, будет 1 1 1 1 1 1 1 1 (шестнадцатеричное FFH).

Удостоверимся в этом:

```
                0000 0001
                1111 1111
  1           - ─────────
(CARRY)         0000 0000
```

1 — перенос

Существует ли способ выработать на основании этого примера общее правило для любого отрицательного числа? Похоже, что нужно взять отрицание числа и добавить единицу в конце.

Испробуем это правило на другом числе, таком, скажем, как 3:

```
  3 =           0 0 0 0  0 0 1 1
1 OPPOSITE      1 1 1 1  1 1 0 0
2 ADD 1 =>      1 1 1 1  1 1 0 1   (FDH)
```

1 — отрицание; 2 — добавлена единица

Давайте сложим это число с 3 и посмотрим, что получится:

```
                0 0 0 0 0 0 1 1
                1 1 1 1 1 1 0 1
(CARRY)-1       ───────────────
                0 0 0 0 0 0 0 0
```

1 - перенос

Получилось!

Мы нашли способ представления отрицательных чисел!

-01 => FF
-02 => FE
-03 => FD и т.д.

Наибольшее положительное число будет

0 1 1 1 1 1 1 1 = 7F =) 127 десятичное

а противоположное ему будет

1 0 0 0 0 0 0 1 = 81 =) -127 десятичное

Настоящей проверкой этого правила будет посмотреть, вернемся ли мы к положительному числу, применив его к отрицательному числу!

Давайте попробуем это на числе -3, для которого выше мы получили значение FDH.

1 NUMBER	1 1 1 1 1 1 0 1
2 OPPOSITE	0 0 0 0 0 0 1 0
3 ADD 1 =>	0 0 0 0 0 0 1 1

1 — число; 2 — его отрицание; 3— добавлено 1

Итак, это представление срабатывает! Мы можем применять его, чтобы получить для любого числа противоположное ему.

16-разрядные отрицательные числа

В точности те же самые рассуждения приложим к числам для двух рук (16-разрядным числам), но только нужно поднимать большой палец одной руки, чтобы показать, отрицательно ли число. (т.е. бит 7 старшего байта).

Условное обозначение

В терминологии ЭВМ это условное обозначение называется «дополнением до двойки». Таблицы значений дополнений до двойки для отрицательных десятичных чисел вы можете найти в приложении к этой книге.

Помните, что это — только условные обозначения! Вам все-таки придется все время решать, должны ли используемые вами числа обозначать числа из диапазона от 0 до 255 или от -128 до +127.

Упражнение:

1. Если 127 (0 1 1 1 1 1 1 1) — наибольшее положительное число, которое можно представить с помощью этих условных обозначений, то как вы представите -128?

2. Найдите наибольшее положительное и наименьшее отрицательное 16-разрядные числа (из двух рук (байтов))?

3. Найдите дополнение до двух наименьшего 16-разрядного отрицательного числа 8000H. Почему это будет 8000H?

Представление алфавитно-цифровых данных

Иногда мы не хотим, чтобы в машинном языке числа представляли команды ЭВМ или были предназначены для вычислений. Мы можем хотеть, чтобы они просто служили символами литер и чисел например, заглавием вашей последней программы, возможно, названной «1-ая в мире программа».

Наша система обозначений алфавитно-цифровых данных, т.е. литер, достаточно прямолинейна: все литеры и числа могут представляться на одной руке (т.е. в 8-разрядном коде).

В мире ЭВМ есть два стандарта представления алфавитно-цифровых литер: код ASCII и код EBCDIC. ASCII означает «американский стандартный код обмена информацией» широко применяется в промышленности микро-ЭВМ. EBCDIC — вариант ASCII, применяемый фирмой IBM. В «ZX SPECTRUM» алфавитно-цифровые литеры удовлетворяют стандарту ASCII, за исключением литер фунт (61H) и авторское право (7H). Таблицу преобразования ASCII вы можете найти в «приложении». Сравните ее с таблицей набора литер в приложении руководства вашего «SPECTRUM».

Попробуйте задать предложение:

PRINT CHR$ 33

и получите в результате (!), поскольку внутренним представлением (!) будет 21H. Помощь!: мы только что показали, что можно говорить, что рука ЦП представляет самые разные вещи:

Это может быть:

Команда ЦП из программы;
Число из диапазона от 0 до 255;
Число из диапазона от -128 до +127;
Часть числа для двух рук;
Алфавитно-цифровая литера.

Все это верно, и это вы, программист, должны помнить, что именно должно храниться на руке ЦП.

Выводы

Содержимое памяти

В памяти «SPECTRUM» могут по желанию храниться программы, числа или текст. Нет никакого способа сказать, что есть что, просто анализируя содержимое отдельной ячейки памяти.

Программы

Команды программ хранятся в памяти в виде последовательностей байтов. Некоторым командам требуется всего один байт. А другим — до четырех байтов.

Числа

Каждая ячейка памяти может использоваться либо для хранения положительных целых чисел, либо целых чисел со знаком (чисел, которые могут быть и положительными и отрицательными) по вашему выбору. Диапазон чисел может быть либо от 0 до 255 либо от -128 до +127.

Отрицательные числа

Было принято соглашение, что когда мы решаем, что в памяти будет храниться число со знаком (+ или -), применяется следующее правило:

Если бит 7 установлен, то число отрицательно.
Если бит 7 не установлен, число положительно. Чтобы получить число, противоположное любому заданному числу, нужно взять «дополнение до двух» и прибавить 1.

Дополнение до 2

Дополнение любого числа до двух — это его отрицание в двоичном формате. Любой установленный бит сбрасывается и наоборот.

Заглянем в ЦП

Введение

Мы говорили, что мозг «SPECTRUM» — ЦП, процессор Z80A. Это — более быстродействующий вариант процессора Z80, производимый по лицензии фирмой «ZILOG».

Единственное отличие между процессорами Z80 и Z80A состоит в том, что первый процессор работает при частоте генератора синхроимпульсов 2 МГц (мегагерц), а второй —

3,5 МГц. «Частота генератора синхроимпульсов» — это просто мера скорости выполнения ЦП вычислений. В «SPECTRUM» за секунду генерируется 3,5 миллиона синхросигналов, т.е. один синхроимпульс каждые 0,000000286 секунды.

Самая быстрая команда, выполняемая ЦП, занимает 4 синхроимпульса, а самая долгая — 21 синхроимульс. Это означает, что даже, если будут выполняться только самые медленные команды, все равно за секунду будет выполняться около 160 000 команд!

Физическая карта мозга

Процессор «SPECTRUM» — кремниевый чип с 40 контактами, пронумерованными от 1 до 40. Эти контакты — линии связи между процессором и остальной ЭВМ. Например, процессор получает питание от источника питания через контакт 11, получает синхроимпульсы через контакт 6, получает и принимает адреса через контакты с 1 по 5 и с 30 по 40 и посылает и принимает данные через контакты с 7 по 15 за исключением 11. Остальные контакты предназначены для передачи сигналов управления.

На этом этапе вам может показаться, что вы полностью запутались. Но не стоит приходить в недоумение, на самом деле наше счастье в том, что мы не знаем внутренней структуры ЭВМ, и нам не нужно ее знать, чтобы пользоваться ее возможностями. Здесь то же самое, что с калькулятором. Физическая структура ЭВМ «прозрачна» для пользователя (иными словами, мы не видим ее!). Нас интересует только логическая структура калькулятора или в данном случае чипа Z80, и как мы можем использовать его для наших целей.

Логическая картина мозга

Логически Z80 можно разделить на пять функциональных частей. Это:

1. Устройство управления.
2. Регистр команд.
3. Счетчик команд.
4. Арифметико-логическое устройство.
5. 24 регистра пользователя (используемые руки и ноги ЦП).

Устройство управления

Мы можем рассматривать устройство управления как некий супервизор, управляющий работой ЦП. Его задача состоит в синхронизации и координации ввода, обработки и вывода для конкретного задания, полученного ЦП вне зависимости от того, исходят ли команды из ПЗУ с записанной в нем программой или из вашей программы.

Регистр команд

Это — рука, используемая ЦП для хранения текущей команды, которую он собирается исполнять. Задание в целом, составляющее программу, должно находиться где-то в памяти — либо в ПЗУ, либо в памяти с произвольным доступом. Вы, может быть, помните, что программа — это последовательность команд. Так, чтобы выполнить задание, устройство управления должно по очереди отыскивать каждую команду в памяти (либо в ПЗУ, либо в памяти с произвольным доступом) и помещать ее в руку, называемую «регистр команд».

Счетчик команд

Это — действительно одна из ног Z80, сообщающая ЦП, где находится следующая часть программы (адрес следующей ячейки памяти, в которой устройство управления должно найти команду). Он подобен указателю для команд, следящему за расположением следующей команды.

Арифметико-логическое устройство (АЛУ)

Это — калькулятор внутри ЦП. Он может выполнять как арифметические, так и логические операции. Из всех основных арифметических функций, известных нам с вами, это устройство может выполнять только простые сложение и вычитание, увеличение (добавление 1) и уменьшение (вычитание 1), но не умножение и деление. Это устройство может также сравнивать числа для одной руки и выполнять «побитовые» операции, такие как перемещение пальцев по кругу, выставление и прижатие определенных пальцев и т.д.

Как побочный результат вычислений, поручаемых АЛУ, обычно меняется состояние различных «флагов» в регистре FLAG. Ниже этот вопрос рассматривается более подробно.

Регистры пользователей

Это — руки и ноги ЦП, которыми вы, программист, можете управлять.

В микропроцессоре Z80 имеется 24 регистра пользователей — некоторые из них будут руки, а некоторые — ноги.

Образ, созданный нами с помощью рук, ног и прямоугольников, позволяет нам легко зрительно представить себе процессы и дает хорошее представление о происходящем, но знатоки программирования будут в недоумении, если вы будете говорить что-то вроде: «... И затем ЭВМ перемещает информацию из правой руки в левую».

Теперь мы дадим вам правильные названия для рук и ног ЦП, так что столкнувшись с подобной ситуацией вы сможете сказать:

«LD B,A»

Во-первых, знатоки программирования называют руки и ноги ЦП «регистрами»

Выше мы говорили, что у ЦП имеется восемь рук: они называются A, B, C, D, E,.... В построенном нами мире рука, по определению, это нечто с 8 пальцами.

У ЦП есть две ноги: они называются IX и IY. По определению, нога — это все что угодно с 16 пальцами!

Названия рук и ног легко запомнить, поскольку название регистра состоит из одной буквы, то это должна быть рука (т.е. он содержит 8 битов), тогда как если название состоит из двух букв, то это будет нога (т.е. он содержит 16 битов).

Вы обратили внимание, как легко мы перешли от пальцев рук и ног к битам? Мы очень быстро приучим вас к терминологии программирования.

На самом деле, остальные две руки ЦП, следующие за D, E... Называются не «G» и «H», как можно было ожидать, а «H» и «L».

Ниже приводится традиционный способ представления всех этих регистров:

A	F
B	C
D	E
H	L
IX	
IY	

Обратите внимание, что «F» соответствует «A», зато порядок остальных достаточно естественен. Причина, по которой регистры сгруппированы таким образом, состоит в том, что иногда можно из двух рук составить ногу!

В конце-концов, если по определению нога — нечто с 16 битами, то возможно, мы можем иногда использовать подделку и применять две 8-битовые руки вместо ноги. Поэтому мы говорим о «парах регистров» таких как BC, DE, и HL.

Причина, по которой пара регистров «HL» называется так, а не «GN», например, состоит в том, чтобы помочь запоминать, в каком из регистров старшее число, а в каком — младшее.

Это аналогично тому, как если бы вы хотели представить числа от 0 до 100 на руках и ногах. Вы легко можете составить из пальцев представление чисел 0 и 10, и аналогично будет с пальцами ног (при условии, что вы достаточно подвижны). Один из способов обозначить таким методом число 37 — отложить 3 на руках и 7 на ногах, но нужно как-то договориться, какое число будет старшим разрядом, а какое младшим, иначе кто-нибудь другой решит, что вы хотели представить число 73.

«H» в «HL» означает «старший» (HIGH), а «L» — младший (LOW), так что нет возможности перепутать, не так ли?

Эта схема пар регистров позволяет также указать, какой из регистров в двух парах регистров содержит старшее число:

B в BC
D в DE

поскольку все старшие и младшие числа размещаются в одинаковом порядке.

Ноги (IX и IY) также имеют специальное название: они называются «индексными регистрами». Это связано во многом с тем, что они применяются для организации информации так, как организуют указатель в книге. С другой стороны, вы можете рассматривать их как указатели таблиц.

Теперь все в порядке, вы поняли терминологию и можно остановиться на некоторых частных вопросах.

Накапливающ й регистр (регистр A)

Это 8-битовый регистр (однобайтовый) — самый важный в Z80. Его название восходит к первым поколениям ЭВМ, когда был лишь один регистр, применявшийся для «накопления» результата.

И хотя мы ушли вперед от первых поколений ЭВМ, накапливающий регистр по-прежнему широко применяется для логических и арифметических операций. На самом деле многие ЭВМ все еще конструируются таким образом, что многие операции можно выполнять только с помощью регистра A.

Это имеет место и для чипа Z80, и регистр A — привилегированный. Вы можете представлять себе регистр A как правую руку ЦП аналогично тому, что многие люди некоторую работу легче выполняют правой рукой, чем левой.

Флаги

Обратите, пожалуйста, внимание, что «AF» обычно не рассматривается как пара регистров. «F» в этом случае применяется для обозначения «регистр флагов». Это — рука с 8 пальцами, так что каждый палец указывает, выполнено или нет определенное условие, и мы вернемся к этому вопросу в отдельной главе.

Пара регистров

Из трех пар регистров (BC, DE, HL) пара HL, возможно, — наиболее важная. Помимо того, что пользователю предоставлен выбор, применять ли ее в виде двух отдельных регистров или как пару регистров, Z80 сконструирован таким образом, что есть определенные

операции 16-битовой арифметики, которые можно выполнять только с помощью пары регистров HL.

Из-за такого привилегированного, с точки зрения аппаратуры положения, операции над парой общих регистров будут обычно быстрее выполняться для регистров HL. Из-за этого HL оказывается предпочтение в программировании на машинном языке.

Возможно, нужно назвать регистр HL правой ногой ЦП?

Альтернативный набор регистров

Я решил, что, возможно, именно здесь следует сказать, что у ЦП есть еще запасной набор рук!

На самом деле скорее не запасной набор рук (ну да, альтернативный набор регистров, если вы хотите придерживаться правильной терминологии), а запасной набор рабочих перчаток...

Это все равно как если бы у вас был набор жестких пластмассовых перчаток, даже столь жестких, чтобы сохранять форму вашей руки, когда вы их снимаете. Если, например, вы отложили на руке число 3 и сняли перчатки, то перчатка все еще будет сохранять форму руки с отложенным числом 3!

Вы, вне всякого сомнения, сразу сообразите, как применить такие перчатки — вы можете запомнить число, пока носите один набор перчаток, сменить перчатки, а старое число останется на случай необходимости в этом другом наборе перчаток!

Другая перчатка находится под рукой, если она вам нужна, и не забудет форму вашей руки, когда вы ее снимали. К сожалению, вы не можете просто взглянуть и увидеть, какое число вы там сохранили. Не может, естественно, также перчатка выполнять какие бы то ни было вычисления, если внутри нее нет руки!

Вам на самом деле придется снова сменить перчатки, чтобы иметь возможность воспользоваться всей той информацией, которую хранят перчатки.

У ЦП есть запасной набор перчаток для каждой пары рук (но не для ног — где это видно, чтобы на ноги надевали перчатки?), но их нельзя переодевать с руки на руку, так же как нельзя на левую руку надеть перчатку с правой руки.

Поэтому теперь представление всех регистров принимает такой вид:

```
A — F  <——>  A' — F'
B — C  <——>  B' — C'
D — E  <——>  D' — E'
H — L  <——>  H' — L'
       IX
       IY
```

Обратите внимание, что набор перчаток, который вы носите, называется так же, как и соответствующая рука, тогда как запасной набор всегда помечается штрихом.

Команды по-прежнему относятся к тому, что делают руки, а не к тому в какой они паре перчаток. Так что, хотя мы и отмечаем запасной набор штрихом, не существует команд, таких как LD A',1. ЦП работает с вашими руками, а не с перчатками

Единственные команды, относящиеся к альтернативному набору регистров - - это команды типа «теперь переоденьте перчатки». Например:

```
1. LD A, (ящик N1)      : загрузить в A содержимое из ящика N1
2. EX AF, AF'           : сокращенная форма «заменить»
                        : (EXCHANGE) — т.е. заменить
3. LD A, (ящик N2)      : перчатки на AF перчатками AF'
4. EX AF, AF'           : еще одна замена
5. LD A, (ящик N3)      :
```

Вы заметите, что среди 5 приведенных выше команд ни одна не оказывает конкретного воздействия на набор альтернативных регистров, однако мы, несомненно, изменили их содержимое.

Этот пример сконструирован для того, чтобы служить иллюстрацией понятия об альтернативном наборе регистров. Постарайтесь понять, что происходит.

Знаете, что будет в регистре «А» после каждой команды? Для простоты предположим, что содержимое трех ящиков таково:

(ящик N1) = 1
(ящик N2) = 2
(ящик N3) = 3

так вот что происходит после каждой команды:

	REGISTER A	REGISTER A'
1.	1	NOT KNOWN
2.	NOT KNOWN	1
3.	2	1
4.	1	2
5.	3	2

REGISTER — регистр; NOT KNOWN — неизвестно.

На самом деле очень просто, не так ли?

Вы обнаружите, что эти заменяющие регистры особенно полезны, когда у вас кончаются руки, кончаются регистры, и вы не хотите освобождать свои руки (ноги), записывая то, что на них отложено в «стек» или в память. Мы рассмотрим этот вопрос позже.

Еще регистры?

Да, есть и еще регистры, но вы, возможно, не будете особенно широко ими пользоваться.

«Указатель стека» — это еще одна нога ЦП (2-байтовый регистр адреса).

Он всегда указывает, до какого размера вырос стек. С ростом стека он смещается вниз к ячейкам памяти с меньшими номерами.

Вам обычно не приходится при программировании на машинном языке делать что-нибудь с указателем стека. За ним следит ЦП и модифицирует его каждый раз, когда вы выполняете PUSH или POP.

Обратите внимание, что часто встречается ошибка, состоящая в том, что забывают вытолкнуть назад значение, которое втолкнули в стек. Можете быть уверены, что это приведет к ситуации «CRASH» (сбой) в вашей программе.

Регистр I

Это — регистр вектора прерывания. В системах, основанных на Z80, отличных от «SPECTRUM» этот регистр обычно применяется для хранения базового адреса таблицы адресов для обработки различных реакций на прерывание, например, запросов ввода-вывода.

Однако, в «SPECTRUM» это средство не применяется, и регистр I участвует в генерации сигналов управления экраном. Вряд ли вам когда-либо придется пользоваться этим регистром.

Регистр R

Регистр R — регистр восстановления памяти. В Z80 он предназначен для автоматического обновления динамической памяти. В процессе работы процессора Z80 информация, хранимая в тех частях динамической памяти, к которым долго не было доступа, будет

«утекать» из-за падения напряжения со временем. Если не обновлять (перезаряжать) эти ячейки памяти, то хранившаяся первоначально информация исчезает!

Регистр R служит простым счетчиком, наращиваемым каждый раз, когда происходит «цикл поиска информации в памяти». Таким образом, значение в регистре R все время циклически меняется от 0 до 255.

Этот факт может использоваться аппаратурой для обеспечения «обновления» всех частей памяти. Не нужно беспокоиться, вам ничего не нужно знать об этом. Дело господина Синклера позаботиться об этом при конструировании «SPECTRUM». Мы можем просто пользоваться его ЭВМ, не думая об обновлении и тому подобных вещах.

С точки зрения программирования вы можете считать регистр R относящимся целиком к аппаратуре и системным применениям. Однако иногда вы можете пользоваться им как средством получения случайного числа от 0 до 255. Это применение мы продемонстрируем ниже.

Выводы

Регистры пользователей

В ЦП есть восемь основных 8-битовых регистров (A, F, B, C, E, H, L) и два 16-битовых регистра (IX, IY). Названия 8-битовых регистров состоят из одной буквы, а 16-битовых — из двух.

Пары регистров

Шесть из восьми 8-битовых регистров могут при определенных условиях использоваться для обработки 16-битовых чисел. Это — пары регистров BC, DE и HL. Название HL может напомнить нам, какой байт старший, а какой — младший.

Предпочтительные регистры

ЦП Z80 сконструирован таким образом, что некоторые 8-битовые команды могут выполняться с помощью регистра A, а некоторые 16-битовые команды — только с помощью пары регистров HL.

Альтернативный набор регистров

Восемь основных 8-битовых регистров можно сменить на другой альтернативный набор регистров.

Значения, хранимые в основных регистрах, сохраняются ЦП, пока используется альтернативный набор, но к ним нельзя осуществлять доступ.

Повторная смена набора регистров позволяет нам вновь обрабатывать первоначальные значения.

Все это хорошо, но как же мне выполнить программу на машинном языке?

Возможно, вы уже достаточно узнали о ЦП и шестнадцатеричной системе счисления, и все это кажется вам не имеющим отношения к делу. Все это не объясняет, как на самом деле выполнить программу на машинном языке. «ZX SPECTRUM» на самом деле все время выполняет программы на машинном языке! (когда он включен). Просто вы об этом не знаете даже когда вы ничего не делаете, просто смотрите на экран, пытаясь придумать, что ввести в качестве первой строки вашей потрясающей основы программы на языке BASIC, ЭВМ «SPECTRUM» занята работой под управлением программы на машинном языке. Это одна из программ, хранимых в ПЗУ, и ее называют «операционной системой». Например, часть программы, выполняющаяся, когда вы просто сидите и смотрите на экран, выполняет следующее: сканирует клавиатуру в поисках введенной информации;

Отмечает, что ни одна клавиша не нажата;
Выдает на дисплей текущий экран (пустой).

Даже когда вы выполняете программу на языке BASIC, ЦП управляется программой на машинном языке. Как мы уже объясняли, эта программа относится к типу «интерпретаторов»: она берет вашу следующую команду языка BASIC, преобразует ее на машинный язык, выполняет эту часть программы и затем возвращается к интерпретации следующей команды.

Все это оказывается не так, когда вы выполняете свою собственную программу на машинном языке!

Полная свобода от операционной системы! Применение функции «USR» передает управление ЦП тем командам, которые вы поместили по адресу USR, каковы бы они ни были. Что бы там ни оказалось, он будет интерпретировать это как допустимые команды машинного языка.

Перспектива довольно пугающая, ведь при потере управления вы можете потерять все, что хранилось в памяти. Одна ошибка, один неверный символ, и вам придется выключать «SPECTRUM» и вновь начинать сначала.

Нет ни сообщений об ошибках, показывающих, где вы ошиблись, ни синтаксической проверки неверных предложений — так что если вы сделаете малейшую ошибку, будут потеряны часы работы, затраченные на ввод вашей программы!

В конце этой книги мы включили программу на языке BASIC, которая позволит вам вводить и редактировать программы на машинном языке. Введя эту программу в ваш «SPECTRUM», запишите ее на ленту, поскольку более чем вероятно, что вы по крайней мере однажды потеряете управление вашей программой на машинном языке.

С другой стороны, не бойтесь экспериментировать — вы не сможете испортить ЭВМ какой бы то ни было программой на машинном языке, вводимой вами. Самое худшее, что может произойти — вам придется выключить и вновь включить ваш «SPECTRUM».

Сейчас мы вам для возбуждения аппетита дадим очень простую программу на машинном языке. Загрузите написанную на языке BASIC программу «редактор машинного языка в коде EZ», найденную в конце этой книги, и выполните ее.

Программа запросит у вас адрес загрузки. Это значит спросит у вас, где бы вы хотели разместить машинные команды. В этой программе кода вы не можете применять адреса менее 31500, так что давайте выберем 32000.

Введите число 32000, затем нажмите (ENTER).

На экране теперь светится: команда или строка (# # #):

Это означает, что программа ждет, чтобы вы ввели команду или новую строку текста машинной программы. Давайте введем «1», затем пробел, затем «C» и затем «9». Это похоже на ввод строки языка BASIC с номером строки1, но это — строка машинного языка. Если все в порядке, то нажмите (ENTER). На экране теперь должны быть показаны все введенные вами строки:

1 C9 и внизу экрана — подсказка «команда или строка (# # #):»

На этом этапе вы не хотите добавлять больше строк, так что давайте вместо этого введем команду.

Введите слово «DUMP» и затем нажмите (ENTER). Действие этой команды состоит в выдаче текста машинной программы в виде распечатки по указанному вами адресу, а именно 32000.

Вас можно поздравить, вы только что ввели команду программы на машинном языке! Вы можете проверить, что она была введена правильно, введя теперь команду «MEM» и затем (ENTER). Эта команда позволяет вам просмотреть содержимое памяти, и она запросит у вас начальный адрес. Введите тогда 32000 и затем (ENTER).

Вы увидите содержимое ячеек памяти с 32000 по 32087. Во всех должно быть 00, за исключением 32000, в которой должно быть C9. Затем нажмите клавишу «M», чтобы вернуться на этап ввода основной команды.

Команда «C9» означает RETURN (возврат)!

Все это несколько напоминает езду на велосипеде в первый раз: вы действительно хотите получить самостоятельность, но как только немного проедете, хочется «вернуться» к надежности твердой земли под ногами (иными словами, операционной системы)

Теперь переходим к выполнению программы на машинном языке. Чтобы выполнить любую программу на машинном языке, записанную вами в память, введите команду «RUN» и затем (ENTER).

Что происходит? Почему внизу экрана появилось число 32000? Это — адрес, использованный вами в качестве адреса загрузки в начале.

Не забывайте, что функция «USR» состоит в выполнении подпрограммы на машинном языке. В частности из этой функции вытекает, что значение USR при возврате из программы на машинном языке, помещенной вами в память, будет равно значению пары регистров BC.

Ответ на этот вопрос вытекает из способа, которым операционная система «SPECTRUM» (да, все та же самая) обращается с функцией «USR».

Когда операционная система встречает функцию «USR», она загружает указанный пользователем адрес в пару регистров BC — в данном случае 32000.

Значение «USR», как в предложении

LET A = USR 32000 естественно, приводит к ответу 32000!

Эта особенность функции «USR» будет очень полезна, поскольку она позволит нам следить за тем, что происходит во время выполнения программы на машинном языке.

Давайте введем следующую программу на машинном языке:

```
0B
C9
```

Ввести эту короткую, состоящую из двух команд, программу можно так.

Ввести строку 1, введя «1», затем пробел, затем «0», затем «B» и затем нажав (ENTER). Аналогичным образом ввести строку 2 C9. Распечатка должна показать вам, что вы ввели строки правильно. Введите команду «DUMP» и затем команду «RUN»

На этот раз результат равен 31999! так получилось потому, что команда «0B» — это «DEC BC» (сокращенное название для уменьшения значения BC на 1).

Упражнение

Поэкспериментируйте с командами, включающими BC, найдя такие команды в таблице в конце книги. Сможете ли вы разобраться, что означают сокращения?

Обратите внимание, чтобы последней строкой во всех ваших программах была «C9». Это — команда RETURN, и если вы ее забудете, программа никогда не совершит возврат.

Если это с вами произойдет, не беспокойтесь — ваша ЭВМ не испортится. Просто выключите питание и загрузите все снова.

Упражнение

Вы можете применять команду «MEM» для проверки любой части памяти. Попробуйте ее для разных адресов, где, как вам кажется, может оказаться что-то любопытное.

Как ЦП применяет свои конечности (регистры).

Введение

Мы видели, что ваш «ZX SPECTRUM» обладает ЦП с 24 руками и ногами. Ключевым вопросом для программирования на машинном языке для вашего «SPECTRUM» будет то, какие операции допускаются и насколько ЦП легко их выполнять.

Вообразите на мгновение, что вы и есть ЦП.

Возможно, как большинство людей, вы правша и есть действия, которые вы более уверенно выполняете правой рукой, чем левой. Могут найтись также такие действия, которые проще выполнять одним способом, а другим — сложнее, например, снять что-нибудь с высокой полки левой ногой и переложить в правую руку сложнее, чем проделать это правой и левой руками.

То же самое относится к машинному языку — вы можете легко выполнять некоторые задания одним способом, другим способом сделать это окажется сложнее, а третьим — вовсе невозможно. Ключ к успеху здесь — в знании, какие комбинации действий допустимы.

Для ЦП эквивалентом вашей правой руки служит регистр «А». Помните? Накапливающий регистр, рука, возникшая в результате генетического родства с ранними ЭВМ.

С другой стороны (простите за каламбур) вы можете временно хранить то, что было в правой руке, в любой другой руке, ноге и наоборот.

Специалисты в области ЭВМ называют это «регистровой адресацией».

Но это — просто официальное название для передачи информации из одного регистра в другой.

Вот еще примеры:

```
LD A,B
LD H,E
```

и так далее.

Обратите, пожалуйста, внимание, что LD — мнемоническое обозначение (сокращение) «LOAD» и что при чтении текста на языке ассемблера операнды переставляются, а запятая читается как «в». Так, предложение

«LD A,B» мы читаем как «загрузить B в A»

Есть и другие комбинации или способы, помимо адресации регистровой, с помощью которых информацию можно передавать из одного регистра в другой, или из регистра в память.

Способы применения конечностей ЦП

Одно из преимуществ процессора Z80 — в большом количестве конечностей и возможных их комбинаций (способов адресации).

Давайте посмотрим, какие комбинации предлагает Z80:

Непосредственная адресация;
Регистровая адресация;
Косвенная регистровая адресация;
Расширенная адресация;
Индексная адресация.

Список названий впечатляет? Не беспокойтесь, сохраняйте уверенность, и мы постепенно пройдем их все одно за другим.

Приведенный выше список не покрывает все возможные комбинации — только те, которые относятся к числам для одной руки! Давайте поочередно рассмотрим каждое из этих возможных видоизменений.

Непосредственная адресация

Общая форма для нее имеет вид:

LD R,N

(или иная команда — мы используем LD только в качестве примера).

Мы используем сокращение «R» для обозначения произвольного 8-битового регистра, а «N» — для 8-битового числа.

Непосредственная адресация — это метод, использующий только одну руку. Реальные данные входят составной частью в команду; это означает, что ЦП может выполнять команду непосредственно после ее получения. Ему не приходится вести поиск дополнительной информации в памяти, чтобы выполнить эту команду.

Например, отложите 215 на руке «А». Я уверен, вы знаете достаточно уже о мнемонике, чтобы суметь написать так:

LD A,215 или LD A,0D7H и опять-таки вы можете сделать то же самое с любым из регистров и любыми числами.

Ниже показан формат команды типа непосредственной адресации.

Байт 1 — код команды (говорящей ЭВМ, что это за команда)
Байт 2 — N (значение реальных данных для команды).

Поскольку под реальные данные отводится один байт, число, которое вы можете указать, ограничено диапазоном 0 — 255. Если вам это не понятно, вернитесь к главе о «способе счета для ЭВМ»

Обычно мы применяем непосредственную адресацию для инициализации счетчиков и определения констант, необходимых для вычислений.

Непосредственную адресацию легко применять в программировании на машинном языке. Тем не менее, это — наименее гибкий из всех методов передачи информации (способов адресации), поскольку и регистр, и данные фиксируются в момент написания программы. Эквивалентное предложение на языке BASIC выглядело бы так: LET A=5. Очевидно, команды такого типа необходимы, но мы не могли бы писать таким способом программы целиком!

Непосредственная адресация удобна, но она не решает основных проблем.

Но мы хотя бы сдвинулись с мертвой точки: как программисты мы теперь можем описать, какие числа и в какие регистры загружаются.

Регистровая адресация

Мы уже кратко рассматривали этот способ. Общий форма имеет вид:

LD R,R

(или другие команды) при этом методе участвуют только две руки; короче, это — передача информации с одной руки на другую.

ЦП допускает передачу информации между любыми двумя руками, за исключением руки «F» (которую вообще не стоит считать рукой). Это регистр «FLAG» и вообще не предназначен для хранения чисел в обычном смысле этого слова).

Для команд с регистровой адресацией требуется всего один байт.

Команды этого типа не только коротки (один байт), они также и быстрее выполняются. Требующееся для их выполнения время составляет 4 импульса генератора синхросигналов, или менее 1 микросекунды для «SPECTRUM».

Для написания программ на машинном языке существует «правило», что для повышения эффективности программы в плане затрат времени и памяти всегда, когда есть возможность, следует применять передачу информации с рук на руки (регистр — регистр).

Косвенная регистровая адресация

LD (RR),A или LD A,(RR)
LD (HL),N

Этот богатый возможностями тип команды приводит к передаче данных между ЦП и ячейкой памяти, на которую указывает содержимое одной из 16-битовых пар (ног).

Косвенная регистровая адресация действует быстрее, чем обычная косвенная адресация, поскольку ЦП нет необходимости извлекать адрес из памяти.

Тем не менее, мы должны сначала загрузить регистр, так что косвенная регистровая адресация дает преимущества только тогда, когда программа использует один и тот же или соседние адреса многократно. например:

```
LD HL,SHAPE          : загрузить начало базы данных SHAPE в HL
LOOP LD A,(HL)       : поиск данных
INC HL               : дать приращение указателю
продолжать LOOP пока не кончится
```

Расширенная адресация

LD A,(NN) или LD (NN),A

Теперь посмотрим, как происходит обмен информацией между памятью и вашими руками и ногами.

При расширенной адресации команда программы предоставляет ЦП адрес, заданный двумя байтами.

Если передача информации осуществляется в или из накапливающего регистра, то она повлияет только на содержимое ячейки памяти, заданной двухбайтовым целым числом.

Если же передача информации осуществляется в или из пары регистров,

то она повлияет и на ячейку памяти, заданную двухбайтовым целым числом, и на следующую ячейку.

Команда такого типа имеет следующий формат:

Байт 1 код операции
Байт 2 (допустимый дополнительный код операции)
Байт 3 младшие разряды 16-битового целого числа
Байт 4 старшие разряды этого целого числа

Именно таким способом программа может считывать содержимое памяти в регистры пользователя. И вновь здесь требуется абсолютный адрес, иными словами, в результате применения этого типа адресации программа может оказаться неперемещаемой, за исключением случаев, когда перемещаемым оказывается абсолютный адрес, на который ссылается команда.

```
SHAPE DB N,N,N,.... : база данных SHAPE
.
.
LD A,(SHAPE) : загрузить первый байт в
накапливающий регистр
```

Индексная адресация

LD R,(IX/IY+Д) или LD (IX/IY+Д),R

(или другие команды)

В передаче информации такого типа участвует нога ЦП, индексный регистр IX или IY.

ЦП складывает содержимое индексного регистра с адресом, заданным в команде, чтобы найти исполнительный адрес.

Это — один из типов команд Z80, имеющих 16-битовый код операции. Другой широко применяемый 16-битовый тип команды — команда загрузки блока, например, LDIR (LOAD INCREMENT AND REPEAT — загрузить, дать приращение и повторить)

Одно из типичных применений метода адресации такого типа — выполнение табличных операций.

Индексный регистр может применяться как указатель на начало таблицы данных. Значение смещения задается в команде, чтобы определить адрес требуемого элемента таблицы, к которому нужно обратиться программе; например:

```
LD IX,TABLESTART     : инициализировать указатель на начало таблицы
LD A,(IX+3)          : обратиться к третьему байту от начала таблицы
```

Команда этого типа имеет следующий формат:

Байт 1 (код операции)
Байт 2 (код операции)
Байт 3 D: целочисленное значение смещения

Число «D» — 8-битовое число, которое необходимо задавать вместе с командой, и оно не может быть переменным. Это означает, что диапазон адресации ограничен значениями от -128 до 127 считая от адреса, заданного индексным регистром.

Индексная адресация выполняется медленнее, поскольку ЦП для получения исполнительного адреса должен выполнить сложение. И все-таки индексная адресация гораздо гибче, поскольку с помощью одной и той же команды можно обрабатывать все элементы массива или таблицы.

Выводы

Есть много способов для ЦП вести поиск 8-битовых порций информации и передачи ее из 8-битовых регистров в память:

Непосредственная адресация
Задает в программе число, передаваемое в любой регистр;

Регистровая адресация
Из любого регистра в любой другой регистр;

Косвенная регистровая адресация
Либо используется BC или DE для задания адреса, а A — для хранения передаваемого числа, Либо же HL для задания адреса и определения числа в программе;

Расширенная адресация
Адрес задается в программе, а A содержит 8-битовое число;

Индексная адресация
IX или IY используются для задания начала таблицы в памяти, а в любом регистре хранится 8-битовое число. Смещение от начала таблицы должно быть задано в программе. Передаваемое в память число также в случае необходимости может быть задано в программе.

Эти способы адресации — единственные способы передачи информации в память и из нее. Никакие другие комбинации не допускаются.

Команды операций загрузки для одной руки

MNEMONIC	BYTES	TIME TAKEN	EFFECT ON FLAGE					
			C	Z	PV	S	N	H
LD REGISTER,REGISTER	1	4	—	—	—	—	—	—
LD REGISTER,NUMBER	2	7	—	—	—	—	—	—
LD A,(ADDRESS)	3	13	—	—	—	—	—	—
LD (ADDRESS),A	3	13	—	—	—	—	—	—
LD REGISTER,(HL)	1	7	—	—	—	—	—	—
LD A,(BC)	1	7	—	—	—	—	—	—
LD A,(DC)	1	7	—	—	—	—	—	—
LD (HL),REGISTER	1	7	—	—	—	—	—	—
LD (BC),A	1	7	—	—	—	—	—	—
LD (DE),A	1	7	—	—	—	—	—	—
LD REGISTER,(IX+Д)	3	19	—	—	—	—	—	—
LD REGISTER,(IY+Д)	3	19	—	—	—	—	—	—
LD (IX+Д),REGISTER	3	19	—	—	—	—	—	—
LD (IY+Д),REGISTER	3	19	—	—	—	—	—	—
LD (HL),NUMBER	2	10	—	—	—	—	—	—
LD (IX+Д),NUMBER	4	19	—	—	—	—	—	—
LD (IY+Д),NUMBER	4	19	—	—	—	—	—	—

MNEMONIC — мнемоническое обозначение; BYTES — количество байтов; TIME TAKEN — затрачиваемое время; EFFECT ON FLOGE — состояние флагов после выполнения; REGISTER — регистр; NUMBER - число; ADDRESS — адрес.

Обозначения для флагов:

\# — указывает, что флаг меняется в результате операции;
0 — указывает, что флаг сброшен;
1 — указывает, что флаг установлен;
– — указывает, что флаг остается неизменным.

Откладывание чисел на одной руке

Поскольку в ЦП «SPECTRUM» все построено на 8-битовых руках и 8-битовых ячейках памяти, очевидно, важно понять, как откладываются числа на руках.

В предыдущей главе мы рассмотрели некоторые способы передачи информации с одной руки на другую. Теперь мы подробнее остановимся на каждом из этих способов. Вы, возможно, помните, что один из них называется регистровой адресацией.

Как мы уже говорили, это просто громкое название для простой передачи информации с одного регистра на другой. Вот примеры:

LD A,B
LD H,E

и т.д.

Напомним используемую терминологию: «LD» означает «загрузить». «,» означает «в» и команда в мнемоническом виде (сокращенном) читается как русское предложение, но с обратным порядком операндов (в оригинале, в соответствии с нормами английского языка говорится о прямом порядке операндов. (примеч. Пер.)).

Таким образом, вслух такую, например, команду, как

LD A,B

мы прочитали бы «загрузить В в А». А следующий пример читался бы как «загрузить Е в Н».

Мы можем передавать информацию с одной руки на любую другую руку, как мы уже говорили. За единственным исключением (регистр флагов отличается от всех других регистров), вы можете задавать передачу с любой руки на любую другую руку. Допускается даже такая, казалось бы глупая команда, как «LD A,A»!

В сокращенной записи это выглядит как «LD R,R», где R представляет собой произвольный 8-битовый регистр, за исключением «F».

Вот и чудно! теперь мы знаем, что умеем перекидывать информацию с руки на руку, но нам от этого мало проку, если на этих руках сначала не будет какой-то информации.

Второй способ, которым мы можем откладывать на руках информацию — это задать ЦП, сколько и на какой руке отложить!

Например, отложить 215 на руке «D». Я уверен, что вам достаточно известно о мнемонике, чтобы вы могли это записать в виде

LD D,D7

(D7 — шестнадцатеричное представление 215)

Возможно, вы помните, что это называется непосредственной адресацией. (название достаточно очевидное, не правда ли?).

Опять-таки, вы можете проделать это с любыми регистрами и любыми числами. Ограничением, конечно, служит размер числа, которое можно задать с помощью 8 битов: от 0 до 256.

Краткая запись такой операции имеет вид «LD R,N», где «R» обозначает произвольный регистр, а «N» — любое число. Здесь по-прежнему действует ранее введенное условное обозначение 8 битов одной буквой.

Вот уже наметились сдвиги: теперь мы можем указать, какие числа в какие регистры загрузить, и можем перекидывать их с руки на руку. Но мы все еще не научились перемещать какие-либо из этих чисел в ячейки памяти, а регистров не так много!

Мы очень кратко показали вам пример «внешней адресации», когда делали упражнение с вычислением разницы во времени:

LD A,(ящик N3).

В виде общего мнемонического обозначения это даст:

LD A,(NN).

Не забудьте, что в нашей краткой записи скобки означают «содержимое».

Обратите здесь внимание на две вещи:

1. Так делать можно только с регистром А.

2. Вы должны задать номер ящика в виде числа для двух рук (16-битового).

Допустима и обратная команда. Это — одна из обращающих на себя внимание особенностей Z80 — в наборе команд имеется симметрия.

LD (NN),A

Обязательно обратите внимание, что эти команды допустимы только для регистра «А». Конечно, есть другие команды для других регистров, но не таких ясных, как эта. Здесь вновь сказывается концепция доминирующей руки.

Давайте задержимся здесь на наносекунду и посмотрим, что эти две команды на самом деле нам дают.

Во-первых, диапазон значений чисел, задаваемых числом для двух рук (NN), будет от 0 до 65 535. Это 64К, то есть с помощью этой команды можно получить доступ не более чем 64К памяти! Это значит, что вся память: ПЗУ, программа, дисплей, свободная память — должна уместиться в 64К. В «16К SPECTRUM» на самом деле 16К используются ПЗУ и 16К — памятью с произвольным доступом, что вместе составляет 32К. 16К относится только к памяти с произвольным доступом. В «48К SPECTRUM» имеются те же самые 16К ПЗУ плюс 48К памяти с произвольным доступом, что вместе составляет 64К. Поэтому для Z80 невозможно получить доступ к большему количеству памяти, чем «48К SPECTRUM».

Команда LD A,(NN) читаемая как «загрузить содержимое ячейки NN в А», — очень богата возможностями. Она позволяет нам читать содержимое любой ячейки памяти, будь то ПЗУ, или память с произвольным доступом.

Вы можете использовать эту команду для исследований по вашему вкусу, даже для тех ячеек, где память отсутствует, например, попытаться посмотреть, что находится за пределами 32К памяти, даже если у вас нет дополнительной памяти. Вас ждет сюрприз — там вовсе не везде нули!

Обратная команда «LD (NN),A», которая читается как — «загрузить А в ячейку памяти NN», будет пытаться вести запись также в любую ячейку памяти, но для нее сказываются физические ограничения.

Вы не можете вести запись в ячейки, которые не могут содержать эту информацию, подобно несуществующей памяти за пределами размера вашей системы.

Одно из свойственных этой команде ограничений состоит в том, что вам необходимо во время написания программы знать, какую ячейку памяти мы хотим проверять или записать в нее.

Сокращение «NN» означает определенное число, например, 17 100, а не переменную.

Вы не можете использовать эту команду в эквиваленте циклу «FOR — NEXT» на машинном языке. Поэтому основное применение этой команды — отведение конкретных ячеек памяти под хранение переменных;

Например, задание

32000 = скорость
32001 = высота
32002 = остаток горючего

в программе типа посадки на Луну.

Поэтому вы могли бы запроектировать программу, в которой вы брали количество оставшегося горючего, уменьшали его и запоминали бы новое количество горючего опять в этой же ячейке.

В момент написания программы вы будете знать адрес той ячейки памяти, которая послужит вместилищем этой информации. (в оригинале: STROEHOUSE это слово отсутствует в двухтомном англо-русском словаре, возможно, опечатка (примеч. Пер.))

Давайте уточним этот момент. Ячейка — не переменная. Это просто ячейка памяти, которую вы используете для хранения информации. Поэтому при написании своей программы на языке ассемблера вы напишите что-нибудь вроде LD A,(FUEL) и когда вы сами или программа ассемблера начнет задавать конкретный текст машинной программы для этой

команды, вы замените «горючее» шестнадцатеричным адресом ячейки памяти, указанной вами.

Но что если мы не знаем точный адрес ячейки памяти, где мы должны искать информацию? Предположим, мы можем только посчитать, где эта информация должна находиться? Поскольку нам необходимо 16 битов, чтобы задать адрес произвольной ячейки памяти, нам придется хранить его в 16-битовом регистре, т.е. в одной из пар регистров BC, DE или HL, или в одном из индексных регистров IX или IY.

Один из способов для этого — сделать так, чтобы в одной из пар регистров содержался адрес ячейки памяти. Поскольку информация содержится в регистре и нам не дан адрес непосредственно, мы называем эту форму адресации косвенной регистровой адресацией.

Мнемоническое сокращение для таких команд имеет вид:

```
LD R,(HL)
LD A,(BC)
LD A,(DE)
```

По-русски эти команды читаются так:
«загрузить в регистр содержимое ячейки памяти, на которую указывает HL»
«загрузить в A содержимое ячейки памяти, на которую указывает BC»
«загрузить в A содержимое ячейки памяти, на которую указывает DE»

Обратите внимание, что, используя «HL» в качестве указателя на нашу ячейку памяти, мы можем провести загрузку в любой регистр, даже в H или L, хотя это и звучит странно, но что, применяя BC или DE, мы можем загрузить только в регистр A. Так происходит потому, что пара регистров HL привилегирована точно так же, как привилегирован регистр A.

И здесь также имеется симметрия по отношению к этим командам. И мы можем аналогичным образом загружать информацию в ячейки памяти:

```
LD (HL),R
LD (BC),A
LD (DE),A
```

Это также называется «косвенной регистровой адресацией» вне зависимости от того, в каком направлении перемещается информация.

С другой стороны, мы могли бы применить индексные регистры IX и IY для указания ячейки памяти. Краткая запись этих команд имеет вид:

```
LD R,(IX + Д)
LD R,(IY + Д)
```

Где «R» — опять произвольный регистр, а «Д» — «смещение» относительно адреса, на который указывает IX или IY. (Не спутайте обозначения «Д» — мы имеем ввиду не регистр «D», а «Д» — смещение (DISPLACEMENT)).

Число «Д» — для одной руки (8-битовое число), которое нужно указывать во время программирования и нельзя сделать переменным. В этом — слабость данной конкретной команды, и это означает, что ее применение обычно ограничивается чтением и записью таблиц, содержащих данные.

Имеется также симметричная команда:

```
LD (IX + Д), R
LD (IY + Д), R
```

Если этот конкретный способ адресации выглядит несколько усложненным, не беспокойтесь, вам он вряд ли понадобится в нескольких первых ваших программах.

Чип Z80 в ЭВМ «Синклер» в высшей степени универсален, и вы можете комбинировать разные способы загрузки чисел, описанных нами выше.

Например, вы можете комбинировать непосредственную адресацию (т.е. задание числа, которое вы хотите загрузить) с внешней адресацией (т.е. задание адреса загрузки с помощью пары регистров).

Это называется (кто бы мог ожидать!) «непосредственной внешней адресацией».

К сожалению вы можете использовать только пару регистров HL и поэтому сокращенная запись имеет вид:

```
LD (HL),N
```

Это удобно, поскольку вы можете непосредственно заполнить ячейку памяти без необходимости сначала загрузить это значение в регистр.

Возможна аналогичная комбинация с индексными регистрами, она называется «непосредственная индексная адресация». Эта команда используется более ограниченно, и сокращенная форма таких команд имеет вид:

```
LD (IX+Д),N
LD (IY+Д),N
```

Применение этих команд в программе на машинном языке

Давайте попробуем на практике применить некоторые из этих команд «LD».

Из предыдущих глав нам известно, что после возврата из «USR» в программе на машинном языке значение «USR» равно содержимому BC. Давайте выполним следующую программу: (сначала загрузите и запустите редактор машинного языка в коде EZ и задайте адрес загрузки 32000)

```
1 0E 00
2 C9
```

Теперь с помощью команды DUMP запишите этот текст в память.

В дальнейшем мы уже не будем давать вам такие явные указания по загрузке и выполнению программ на машинном языке, поскольку такой метод трудоемок и не дает вам никакого дополнительного понимания сути программы.

Мы будем предполагать, что к настоящему моменту вы достаточно знакомы с написанном на «Бейсике» «редакторе машинного языка в коде EZ» и с помощью таблиц в конце книги сможете ввести программу. Поэтому все наши программы мы будем приводить в следующем виде:

```
OE 00 LD C,0
C9 RET
```

В этих обозначениях текст программы на машинном языке приводится слева, а мнемонические обозначения ассемблера — в колонке справа. В них также очень ясно показано, для каких команд требуется всего 1 байт (такие как RETURN), каких — 2 байта и т.д. (напомним, что в Z80 некоторые команды могут занимать до 4 байтов!)

Еще одна проблема состоит в том, что мы попробуем сделать все наши программы независимыми от начального адреса (где начинается программа в памяти), так что не имеет значения, что вы укажете в качестве адреса начальной загрузки.

Тем не менее помните, что эти программы можно вводить с помощью программы «редактор машинного языка в коде EZ», помещенной в конце книги, или любой другой программы загрузки, которую вы сами можете разработать.

Какого результата вы ожидаете до выполнения этой программы на машинном языке (вы должны «ввести» текст программы в память и затем применить команду «RUN» в программе кода EZ)?

Программа устанавливает регистр «C» в паре регистров BC равным нулю, а вы знаете, что BC начинается с адреса программы, равного 32 000. Пусть предполагаемый ответ будет

A. 0000
B. 32000
C. 31896

Теперь выполните программу. Совпадет ли ответ с тем, что вы предполагали? Если вам неясно, почему получился такой ответ, вернитесь и перечитайте главу «как считает ЭВМ». Теперь попытайтесь выполнить следующую программу:

```
06 00 LD B,0
0E 00 LD C,0
C9 RET
```

Это даст вам ожидаемый результат 0 при BC=0 (оба регистра в и с установлены равными 0).

Упражнение

Вам может доставить удовольствие попробовать решить несколько занимательных задач, таких как загрузить какое-то число в A, переместить его в L, установить H равным 0 и тому подобное.

Упражнение

Файл атрибутов начинается с адреса 5800H. Мы можем установить HL так, чтобы указывать на файл атрибутов, с помощью следующей программы:

```
26 58 LD H,58H
2E 00 LD L,0
```

Это означает, что вы можете теперь менять цвета на дисплее с помощью команды LD (HL),N.

Структура файла атрибутов описана в руководстве по «SPECTRUM». Давайте установим первую литеру так: бумага красная, чернила белые, режим мигания включен.

10111010=BAH,

так что следующая строка программы будет такая:

```
36 BA LD (HL),BAH
```

Далее, вы никогда не должны забывать вернуться из программы на машинном языке, так что последней строкой должна быть:

```
C9 RET
```

Выполните эту машинную программу. Работает ли она?

Флаги и их применение

Флаги — это такие красивые флажки, которыми вы размахиваете на государственных праздниках..... — вовсе нет!

В машинном языке «флаг» означает «отметка». Флаг — это то, что вы поднимаете, если хотите пометить для кого-то другого, что выполнено определенное условие.

Очевидна аналогия с мореплаванием, где флаги поднимают в знак траура, для обозначения страны, пиратского корабля или еще чего-нибудь.

Причина, по которой конструкторы Z80 (как и большинство конструкторов ЦП) используют флаги в своих машинных языках — в том, чтобы дать программисту информацию о числе, находящемся на доминирующей руке ЦП (в регистре «A»), или информацию о только что выполненно последнем вычислении.

Напомним, что один из регистров ЦП отведен под флаги — это регистр «F». Вы могли также обратить внимание на то, что в начале последней главы в таблице сведены различные команды, которые должны были в этой главе рассматриваться, и часть таблицы была отведена на описание воздействия каждой команды на флаги. (К счастью, ни одна из команд этой последней главы не влияла ни на один флаг.)

Легче всего понять функции флага нуля.

Этот флаг будет подниматься на флагшток, если содержимое регистра «А» равно нулю. Есть много важных решений, которые будут зависеть от того, равен ли «А» нулю. Обратите внимание, что флаг нуля либо установлен, либо сброшен. Вы не можете получить промежуточный результат (в стиле «немножко беременна»), так что вам понадобится всего один бит для определения флага нуля.

То же самое относится ко всем остальным флагам. Они либо установлены, либо сброшены и требуют всего одного бита.

Другие типы флагов

Регистр «F» — обычный 8-битовый регистр, и поэтому в нем можно разместить 8 различных флагов. На практике, однако, конструкторы придумали всего 6 флагов!

S	1. SIGN FLAG
Z	2. ZERO FLAG
H	3. HALF-CARRY FLAG
P/V	4. PARITY FLAG
	5. OVERFLOW FLAG
N	6. SUBTRACT FLAG
C	7. CARRY FLAG

1 — флаг знака; 2 — флаг нуля; 3 — флаг половинного переноса 4 — флаг четности; 5 — флаг переполнения; 6 — флаг вычитания; 7 — флаг переноса

На самом деле разработчики придумали семь флагов, но решили, что одним регистром (так в оригинале. Очевидно, имеется в виду бит. (примеч. Пер.)) можно реализовать два флага: и четности и переполнения.

Давайте теперь подробнее рассмотрим каждый из этих флагов.

Флаг нуля

Этот флаг мы уже рассматривали выше. Его прикладное значение очевидно, он обычно устанавливается после арифметических операций, поскольку служит для описания содержимого регистра «А».

Обратите, однако, особое внимание на то, что в регистре «А» может содержаться 0, а флаг нуля установлен не будет. Так легко может случиться при выполнении команды

```
LD A,0
```

Мы выше уже говорили, что ни одна из команд загрузки для одной руки (8-битовая) не оказывает никакого влияния ни на один из флагов. Флаг нуля не будет установлен, а в А будет ноль.

Флаг нуля также устанавливается, если нулевым оказывается результат команд из группы «перемещений в круговом порядке и сдвигов».

Кроме того, флаг нуля оказывается единственным зримым результатом некоторых команд проверки, таких как группа команд «проверка бита». В этих случаях флаг нуля устанавливается, если проверяемый бит равен нулю.

Флаг знака

Флаг знака очень напоминает флаг нуля и действует по преимуществу на том же самом наборе команд (расхождение в основном касается группы «проверки бита», где понятие отрицательного бита при любой интерпретации лишено смысла).

Флаг переноса

Это — один из наиболее важных флагов, имеющихся в языке ассемблера, поскольку без него результаты арифметических действий в ассемблере были бы совершенно бессмысленными.

Что нужно помнить, это то, что команды языка ассемблера всегда относятся либо к числам для одной руки (8-битовым) или к числам для двух рук (16-битовым).

Это означает, что числа, с которыми мы имеем дело, могут быть либо:

8-BIT —> 0-255
16-BIT —> 0-65535

либо, если вы включаете перенос:

8-BIT —> 0-256
16-BIT —> 0-65536

рассмотрим ситуацию, в которой мы выполняем следующее вычитание

```
   200
 -
   201
 ----
Результат = 255 !!!
```

Это — прямое следствие того, что у нас имеется только ограниченный диапазон чисел, и то же самое может произойти, очевидно, и с 16-битовыми числами.

Мы уже видели, что на одной руке вы можете считать только до 255. Что произойдет, если регистр уже содержит 255, и вы добавляете 1? Возможно, вам понравится представлять себе регистр работающим так же, как спидометр в вашем автомобиле. После того, как вы достигли максимума, он «проворачивается» и снова начинает отсчет от нуля.

Точно так же, если в регистре или на спидометре автомобиля все нули, и вы поворачиваете его назад, вам высветится наивысшее значение, или для 8-битового регистра — 255.

Именно поэтому результат 200-201 дает 255. Если бы были торговцами машинами, мы очевидно захотели бы иметь какой-то знак, что спидометр прокрутился вперед (в этом случае автомобиль прошел больше, чем кажется) или назад (в этом случае его показания подделаны).

Такого типа указатели имеются при программировании на машинном языке, и их называют флагами переноса. К счастью, нам нет нужды беспокоиться о подделке значения регистра. Мы видели, что флаг переноса может быть установлен в результате вычитания при наличии «ухода за ноль». Флаг переноса может быть также установлен в результате операций сложения если должно быть «переполнение». Поэтому удобно представлять себе бит переноса как 9-й бит регистра «А»:

NUMBER	CARRY BIT	NUMBER IN BIT FORM
132	—	10000100
+ 135	—	10000111
267	1	00001011

NUMBER — число; CARRY BIT — бит переноса; NUMBER IN BIT FROM число в виде битов.

Однако поскольку у нас нет 9 битов, а регистре «А» будет содержаться число 0BH (десятичное — 11) и перенос будет установлен (т.е. равен 1).

Можно заметить, что при вычитании занимание из 9-го бита также приведет к появлению в нем «1».

Использование флагов в конструкции на машинном языке, эквивалентной предложению «IF... THEN,...»

В языке BASIC у нас есть возможность конструировать ситуацию типа «IF... THEN», такие как

```
IF A=0 THEN .....
1. WHERE WHAT FOOLLOWS CAN BE "LET...."
                              2   OR "GOTO...."
                              3   OR "GOSUB..."
```

1 — где-то, что следует, может быть; 2 — или принятия решения точно такого же типа можно запрограммировать на машинном языке (за исключением предложения «LET... ») вместо того чтобы написать «IF A=0», мы просто проверяем флаг нуля: если он установлен, то мы знаем, что A=0. Три рассмотренных нами к настоящему моменту флага — это в основном все, что нам нужно, чтобы выбрать следующую команду для выполнения.

Подобная команда имеет следующий формат:

```
JP CC, END,
```

где «JP» — мнемоническое обозначение команды перехода (JUMP), а «END» — удобная метка.

Эта команда по-русски читается как «перейти при условии CC на END» условие «CC» может быть любым из следующих:

Z	(= ноль)
NZ	(= не ноль)
P	(= положительное)
M	(= минус)
C	(= перенос установлен)
NC	(= нет переноса)

Флаг четности/переполнения

Этот флаг для некоторых команд действует как флаг четности, а для других — как флаг переполнения, однако путаница возникает редко, поскольку эти два типа команд обычно не встречаются вместе.

Роль флага как указателя четности выступает в логических операциях, и он устанавливается в том случае, если в результате установленным оказывается четное количество битов. Более подробно этот вопрос мы рассматриваем в главе логических операциях.

Переполнение говорит вам о том, что только что выполненная вами арифметическая операция может дать результат, не умещающийся в 8 битов. Вместо того, чтобы на самом деле означать, что для записи результата понадобился 9-й бит, этот флаг говорит о том, что в результате операции 8-й бит изменился!

В приведенном выше примере до сложения 132 и 135 8-й бит был равен «1», а после — «0», так что должен был быть установлен флаг переполнения. Но флаг переполнения будет также установлен и в результате такого сложения:

```
 64      01000000
+65      01000001
```

129	10000001

Флаг вычитания

Этот флаг устанавливается, если последней операцией было вычитание!

Флаг половинного переноса

Этот флаг устанавливается способом, аналогичным флагу переноса, но только в том случае, когда имеется переполнение или занимание из 5-го бита, а не из 9-го!

Как флаг вычитания, так и флаг половинного переноса применяются только для арифметических операций над «двоично кодированными десятичными числами», мы рассматриваем эти флаги в главе «арифметические действия над двоично-кодированными десятичными числами».

Выводы

Флаги применяются ЦП для указания определенных условий после выполнения команд.

Имеется шесть таких флагов, о каждом из которых можно говорить, что он установлен или сброшен. Представляющие эти флаги биты — это шесть из восьми битов регистра F. Два остальных бита не используются.

Различными флагами отмечаются следующие ситуации:

Перенос; Ноль; Четность или переполнение; Знак; Отрицание; Половинный перенос

Не все команды оказывают влияние на все флаги сразу. Некоторые влияют на все флаги, а некоторые — только на конкретные, другие же вовсе не оказывают на них действия.

Увеличение и уменьшение чисел на одной и двух руках

В последней главе мы рассмотрели понятие флага, а в предшествующей ей — выяснили, как ЦП может загружать любые нужные числа на свои руки и ноги. Давайте теперь рассмотрим простейший способ манипулирования числами на пальцах: мы можем увеличивать или уменьшать представленные числа.

Это — достаточно элементарные арифметические действия. Но это все-таки сложнее, чем просто откладывать конкретные числа на пальцах. Действие увеличения по существу сводится к увеличению на единицу любого числа, отложенного на пальцах.

Это действие можно использовать в таких обычных ситуациях, как перепись населения или управление движением на некотором перекрестке.

Увеличение

У Z80 имеется возможность увеличить число, положенное на пальцах любой одинарной руки, имеющейся у ЦП. Именно это подразумевается общим мнемоническим обозначением:

```
INC R
```

«INC» по-английски читается как «INCREASE» (увеличить) и говорит само за себя.

Можно также увеличит число, находящееся на любой ноге (включая пары регистров, которые, на самом деле, ногами не являются, как мы уже видели).

Это увеличение числа на наших ногах записываются так:

```
INC RR
INC IX
INC IY
```

где «RR» обозначает пару регистров, такую как «BC», «DE», «HL»

Обратите еще раз внимание на простой способ обозначения того, какие операции используют 8-битовые числа, а какие 16-битовые. 8-битовые числа обозначаются одной буквой, а 16-битовые — двумя.

Однако команда «увеличения» на самом деле имеет еще большие возможности, чем видно из приведенных примеров. Можно увеличить содержимое любой ячейки памяти, если мы сможем задать ее адрес с помощью индексных регистров или привилегированной пары регистров, HL:

```
INC (IX+Д)
INC (IY+Д)
INC (HL)
```

где «Д» — смещение (DISPLACEMENT), а не регистр D!

Важное замечание

Вспомним поточнее, как мы условились понимать употребление скобок: скобки означают — «содержимое». Это очень важно, поскольку есть большое сходство между командами

```
INC HL
INC (HL),
```

но при выполнении они очень сильно различаются.

Первая читается как «увеличить HL», тогда как вторая «увеличить содержимое ячейки, адресом которой является HL». (этот второй способ чтения часто сокращается до «увеличить содержимое HL»).

Если только вы помните правила мнемонических сокращений, вы избегнете путаницы такого рода. Давайте посмотрим, как работает каждая из этих команд в предположении, что HL=5800H.

```
INC HL
```

Проверить HL. Увеличить на единицу отложенное на его пальцах число. Результат: HL = 5801H

```
INC (HL)
```

Проверить HL. Найти ячейку памяти, на которую ссылается это число. Увеличить число в этой ячейке на единицу. Результат: HL = 5800H. (5800H) = (5800H)+1. У этих операций есть существенные различия. (Вы можете захотеть выполнить обе версии — 5800H — это начало файла атрибутов). Обратите также ваше внимание на то, что «INC HL» — команда, действующая на 16-битовое число, тогда как «INC (HL)» — команда, действующая только на 8-битовое число, на число, хранимое в ячейке 5800H!

Уменьшение числа

Симметричность набора команд Z80 почти наверняка должна обеспечивать, чтобы все, что вы можете уменьшить, вы могли также и увеличить, и именно так обстоит дело:

```
DEC R
DEC RR
DEC IX
```

DEC IY
DEC (HL)
DEC (IX+Д)

DEC (IY+ Д) мнемоническое сокращение «DEC» по-английски читается как «DECREASE» (уменьшить), и здесь нужно тоже быть внимательным при использовании скобок.

Влияние на флаги

Поскольку команды уменьшения и увеличения 8-битовых чисел воздействуют на все флаги, за исключением флага переноса, именно здесь удобно дать обзор работы с флагами.

Важное замечание: команды увеличения и уменьшения для 16-битовых чисел не влияют на какие бы то ни было флаги. Изменяют флаги только операции увеличения и уменьшения для 8-битовых чисел.

Флаг знака: этот флаг будет установлен в (=1), если бит 7 8-битового результата равен 1.

В соответствии с нашей предшествующей аналогией это означает, что он будет установлен, если поднят большой палец. Обратите внимание, что это произойдет вне зависимости от того, какой способ представления чисел используется.

Флаг нуля: этот флаг будет установлен (=1), если 8-битовый результат равен нулю.

Флаг переполнения: этот флаг будет установлен (=1), если содержимое бита 7 8-битового числа изменяется в результате операции.

Флаг половинного переноса: этот флаг будет установлен (=1), если имеется перенос или занимание из четвертого бита 8-битового числа.

Флаг отрицания: этот флаг устанавливается, если последней выполненной командой было вычитание. Так, он сбрасывается (=0) для «INC» и устанавливается (=1) для «DEC».

Предлагаемые упражнения

С помощью группы команд «LD», «INC» и «DEC» сделайте так, чтобы в результате операции «USR» возвращались нужные вам числа. Это позволит вам освоиться с этими командами.

Выводы

Мы можем увеличивать или уменьшать содержимое в любом из 8-битовых регистров, в любой из 16-битовых пар регистров или в любом из 16-битовых индексных регистров.

Мы можем также увеличивать или уменьшать содержимое ячеек памяти, адреса которых заданы парой регистров HL или индексными регистрами.

Увеличение или уменьшение 16-битовых чисел не оказывает влияния ни на один из флагов. Увеличение или уменьшение 8-битовых чисел как в регистрах, так и в памяти влияет на все флаги, за исключением флага переноса.

Арифметические операции для одной руки

По поводу арифметических операций для одной руки нужно только заметить, что все операции в данной главе относятся только к 8-битовым числам и все они должны выполняться с помощью привилегированной руки, регистра А.

Возникает впечатление, что только наша привилегированная рука умеет складывать и вычитать!

Этот факт настолько органично вплетается в мнемонику машинного языка Z80, что в некоторых мнемонических обозначениях сокращение «А» просто опускается. Например, если нужно вычесть «В» из «А», естественно было бы ожидать команды, подобной

SUB A,B

На самом же деле мнемоническое обозначение имеет вид:

SUB B

Команды арифметических операций для одной руки

MNEMONIC	BYTES	TIME TAKEN	EFFECT ON FLAGS					
			C	Z	PV	S	N	H
ADD A,REGISTER	1	4	#	#	#	#	0	#
ADD A,NUMBER	2	7	#	#	#	#	0	#
ADD A,(HL)	1	7	#	#	#	#	0	#
ADD A,(IX+Д)	3	19	#	#	#	#	0	#
ADD A,(IY+Д)	3	19	#	#	#	#	0	#
ADC A,REGISTR	1	4	#	#	#	#	0	#
ADC A,NUMBER	2	7	#	#	#	#	0	#
ADC A,(HL)	1	7	#	#	#	#	0	#
ADC A,(IX+Д)	3	19	#	#	#	#	0	#
ADC A,(IY+Д)	3	19	#	#	#	#	0	#
SUB A,REGISTR	1	4	#	#	#	#	1	#
SUB A,NUMBER	2	7	#	#	#	#	1	#
SUB A,(HL)	1	7	#	#	#	#	1	#
SUB A,(IX+Д)	3	19	#	#	#	#	1	#
SUB A,(IY+Д)	3	19	#	#	#	#	1	#
SBC A,REGISTER	1	4	#	#	#	#	1	#
SBC A,NUMBER	2	7	#	#	#	#	1	#
SBC A,(HL)	1	7	#	#	#	#	1	#
SBC A,(IX+Д)	3	19	#	#	#	#	1	#
SBC A,(IY+Д)	3	19	#	#	#	#	1	#
CP REGISTER	1	4	#	#	#	#	1	#
CP NUMBER	2	7	#	#	#	#	1	#
CP (HL)	1	7	#	#	#	#	1	#
CP (IX+Д)	3	19	#	#	#	#	1	#
CP (IY+Д)	3	19	#	#	#	#	1	#

MNEMONIC — мнемоническое обозначение; BYTES — байты; TIME TAKEN — время выполнения; EFFECT ON FLAGS — воздействие на флаги; REGISTER — регистр; NUMBER — число.

Обозначения флагов:

#	показывает, что флаг изменен операцией;
0	показывает, что флаг сбрасывается;
1	показывает, что флаг устанавливается;
–	показывает, что флаг остается неизменным.

Вопреки этим ограничениям, накладываемым на арифметические команды (они должны ограничиваться регистром A), язык Z80 очень разносторонен в плане того, что мы можем сложить с произвольным числом, отложенным на привилегированной руке:

ADD A,R	сложить с A произвольный одинарный регистр;
ADD A,N	сложить с A произвольное 8-битовое число;
ADD A,(HL)	прибавить 8-битовое число, находящееся в ящике, адрес которого задается (HL);
ADD A,(IX+Д)	прибавить 8-битовое число, находящееся в ящике, адрес которого задается (IX+Д);
ADD A,(IY+Д)	прибавить 8-битовое число, находящееся в ящике, адрес которого задается (IY+Д).

Вы можете оценить чрезвычайно широкий диапазон возможных чисел, которые мы можем сложить с произвольным числом, хранимым в A: любое число, любой регистр и практически произвольный способ, которым нам придет в голову определить ячейку памяти.

Отсутствует только следующий:

ADD A,(NN) в котором мы определяем адрес в ходе выполнения программы. В результате единственный способ получить такую команду, это написать:

```
LD HL,NN
ADD A,(HL)
```

Обратите также внимание, что регистр HL вновь играет привилегированную роль. Мы не можем задавать ячейки памяти с помощью пар регистров BC или DE.

Другое ограничение, неявно присутствующее во всех этих командах — неустранимое ограничение 8-битовыми числами, которые могут содержать Только значения до 255, как мы уже видели.

Например, такие команды:

```
LD A,80H
ADD A,81H
```

дадут в результате в «A» только 1, но флаг переноса будет установлен, чтобы отметить, что результат не уместился.

Если вас смущает шестнадцатеричная арифметика, то хорошо для практики преобразовать числа в десятичную систему и проверить сложение.

Шестнадцатеричное сложение и вычитание такие же, как и в обыкновенной арифметике:

```
1 + 1 = 2
1 + 2 = 3
```

и т.д., но когда вы доходите до 1 + 9, получается

```
1 + 9 = A
1 + A = B
```

и т.д., а когда вы доходите до 1 + F, получается

1 + F = 10

это происходит потому, что перенос происходит в следующий столбец, когда у вас получается число, превышающее «F», а не «9», как в десятичной арифметике.

Результат нашей приведенной выше программы на машинном языке поэтому будет таков:

```
 80
+81
---
101H   так как 8 + 8 = 16 ==> 10H
```

Что можно поделать с этой ошибкой при переносе?

Конструкторы Z80 предоставили нам еще одну команду, аналогичную ADD, но принимающую во внимание возможность переполнения в флаге переноса.

Это — очень полезная команда: «ADC», читающаяся как «сложение с переносом» (ADD WITH CARRY).

Она совершенно совпадает с командой «ADD», имеет тот же диапазон чисел, регистров, и т.п... Которые можно складывать с регистром «A», за исключением того, что перенос также складывается (если он установлен).

Это дает возможность складывать числа, превышающие 255, с помощью цепочки операций:

Например, сложить 1000 (т.е. 03E8H) с 2000 (т.е. 07D0H) и запомнить результат в BC:

```
LD A,E8H     : младшие разряды первого числа
ADD A,D0H    : младшие разряды второго числа
LD C,A       : записать результат в C
LD A,03H     : старшие разряды первого числа
ADC A,07H    : старшие разряды второго числа
LD B,A       : записать результат в B
```

После первого сложения (E8 +D0) у нас флаг переноса будет установлен (поскольку результат был больше FF), а в регистре a будет B8 (проверьте это самостоятельно).

Второе сложение (3+7) даст не 0AH (= 10 десятичное), как может показаться на первый взгляд, а 0BH (= 11 десятичное) из-за переноса.

Поэтому конечный результат будет равен 0BB8H=3000! Такую цепочку можно продолжить и обработать число произвольного размера, а результат будет храниться в памяти, а не в паре регистров.

8-битовое вычитание

Оно совершенно такое же, как и 8-битовое сложение. Имеется два набора команд, один для обычного вычитания, другой — для вычитания с переносом:

```
SUB S  — вычесть S
SBC S  — вычесть S с переносом.
```

Обозначение «S» предназначено для того же диапазона возможных операндов, что и для команд сложения.

Сравнение двух 8-битовых чисел

Давайте отвлечемся на минутку от машинного языка и посмотрим, что же мы на самом деле понимаем под сравнением двух чисел:

Мы знаем, что происходит, когда два сравниваемых числа совпадают — они «равны». Один из способов обозначения этого факта в арифметической форме — сказать, что разность между двумя числами равна нулю.

Что, если сравниваемое число превышает первое число (сравнение предполагает отношение между двумя числами: мы сравниваем число с тем, что у нас уже было отложено на пальцах)? Тогда результат после вычитания нового числа будет отрицательным.

Аналогично, если новое число было меньше, то разность будет положительной.

Мы можем с помощью этих понятий разработать систему сравнения на машинном языке. Все, что нам нужно — это флаги и операция вычитания. Предположим, нам нужно сравнить числа из некоторого диапазона с числом, скажем 5:

```
LD A,5      : имеющееся число
SUB N       : сравниваемое число
```

Тогда у нас получится следующие результаты: если N = 5 флаг нуля уставлен, флаг переноса сброшен. Если N < 5 флаг нуля сброшен, флаг переноса сброшен. Если N > 5 флаг нуля сброшен, флаг переноса установлен.

Поэтому ясно, что проверку на равенство дает флаг нуля, а проверку на «превышение» — флаг переноса. (проверкой на «меньше» будет сброс обоих флагов).

Единственное неудобство этого метода состоит в том, что содержимое регистра «А» изменяется в результате этой операции.

К счастью у нас есть операция «CPS». По-английски она читается как «COMPARE» (сравнить). Обратите внимание, что она позволяет сравнивать только то, что у нас уже имеется в регистре «А»: диапазон возможных чисел для сравнения такой же, как и для сложения. «Сравнение» совершенно совпадает с «вычитанием» за исключением того, что содержимое регистра «А» не изменяется. Результат сказывается таким образом только на флагах.

Выводы

Восьмибитовые арифметические операции для Z80 ограничиваются:

Сложением; Вычитанием; Сравнением; и могут выполняться только с помощью регистра «А».

При этих ограничениях, однако, имеется широкий диапазон режимов адресации.

Из-за присущих 8-битовым числам свойств мы все время должны тщательно следить за переполнением. Флаг переполнения (так же как и другие флаги) изменяется в результате арифметических операций. Мы можем пользоваться этим как предупреждением о возможном переполнении.

Дополнительные команды (сложение с переносом и вычитание с переносом) позволяют нам выполнять арифметические операции цепочкой, чтобы обрабатывать переполнение.

Логические операторы

Есть три операции, которые в области программирования на машинном языке (или языке ассемблера) имеют такое же значение, как и более широко используемые сложение, вычитание, умножение и деление в обычной арифметике.

Их обычно называют булевскими операторами в честь человека, сформулировавшего для этих операций правила. Это следующие операции:

```
AND
OR
XOR
```

Мы уже знакомы с понятием операций, приложимых ко всему 8-битовому числу, но причина, по которой эти операции имеют такое значение, состоит в том, что они затрагивают отдельные биты числа (или пальцы на руке ЦП). Давайте рассмотрим одну из этих операций «AND»:

BIT A	BIT B	RESULT OF BIT A «AND» BIT B
0	0	0
1	0	0
0	1	0
1	1	1

BIT — бит, RESULT — результат

Очевидно, что результат операции «AND» состоит в том, что «1» получается только в том случае, если «A» и «B» оба содержали «1».

На машинном языке, если вы выполняете AND для двух чисел, то результат будет таким, как если бы вы выполняли «AND» для каждого отдельного бита этих двух чисел.

Вы можете задать себе вопрос: «в чем смысл такой операции?»

Операция «AND» очень полезна тем, что позволяет нам маскировать байт, так что он изменяется так, чтобы содержать только определенные биты.

Если, например, мы хотим ограничить конкретную переменную диапазоном 0 — 7, нам хочется вполне ясно указать, что информация должна содержаться только в битах 0 — 2. (если бы информация содержалась в бите 3, то число было бы равно по крайней мере 8). Например:

0 0 0 0 0 1 0 1 = 5
(————————)

Эти биты должны быть «0»

Поэтому если мы берем некоторое число, значение которого нам неизвестно, и совершаем над ним операцию «AND» с «7», то в результате получится число в диапазоне 0—7. Например:

	01101001	= 105	
	00000111	= 7	=> маска
результат операции «AND»	00000001	= 1	=> в диапазоне 0—7

Команды логических операций

MNEMONIC	BYTES	TIME TAKEN	EFFECT ON FLAGS					
			C	Z	PV	S	N	H
AND REGISTER	1	4	0	#	#	#	0	1
AND NUMBER	2	7	0	#	#	#	0	1
AND (HL)	1	7	0	#	#	#	0	1
AND (IX+Д)	3	19	0	#	#	#	0	1
AND (IY+Д)	3	19	0	#	#	#	0	1
OR REGISTER	1	4	0	#	#	#	0	0
OR NUMBER	2	7	0	#	#	#	0	0
OR (HL)	1	7	0	#	#	#	0	0
OR (IX+Д)	3	19	0	#	#	#	0	0

MNEMONIC	BYTES	TIME TAKEN	EFFECT ON FLAGS					
			C	Z	PV	S	N	H
OR (IY+Д)	3	19	0	#	#	#	0	0
XOR REGISTER	1	4	0	#	#	#	0	0
XOR NUMBER	2	7	0	#	#	#	0	0
XOR (HL)	1	7	0	#	#	#	0	0
XOR (IX+Д)	3	19	0	#	#	#	0	0
XOR (IY+Д)	3	19	0	#	#	#	0	0

MNEMONIC — мнемоническое обозначение; BYTES — байты; TIME TAKEN — время выполнения; EFFECT ON FLAGS — воздействие на флаги; REGISTER — регистр; NUMBER — число.

Обозначения флагов:

— означает, что флаг изменен операцией;
0 — означает, что флаг сбрасывается;
1 — означает, что флаг устанавливается;
– — означает, что флаг не изменился.

Обратите внимание, что чип Z80 позволяет выполнять операцию «AND» только с регистром «A». Для регистра «A» можно выполнить операцию «AND» с 8-битовым числом, любым из остальных 8-битовых регистров, (HL), (IX+) или (IY+);

Например:

```
AND 7
AND E
AND (HL)
```

Обратите внимание, что поскольку действие определено только для регистра «A», нет необходимости упоминать его в команде.

Для других булевских операций, «OR» и «XOR», имеет место тот же диапазон возможностей и ограничение использования регистром «A».

Операция «OR» по смыслу аналогична операции «AND»:

BIT A	BIT B	BIT A OR BIT B
0	0	0
0	1	1
1	0	1
1	1	1

Очевидно, результат операции «OR» должен давать нам «1», если «A» или «B» содержало «1».

Вновь вы можете задать вопрос, в чем смысл такой операции.

Операция «OR» также весьма полезна тем, что она позволяет нам устанавливать любой бит числа, если, например, нам желательно гарантировать, что число будет нечетным, то совершенно очевидно, что нам нужно установить бит 0. (того же результата можно добиться с помощью команды «SET»).

```
LD A,NUMBER
OR 1                    ; MAKE NUMBER ODD NUMBER — число;
```

MAKE NIMBER ODD — сделать число нечетным.

Приведенные выше две строки — типичный фрагмент распечатки на ассемблере.

Смысл операции «XOR» (произносится «исключающее или») также легко понять, но ее реальное использование в программировании более ограничено. Результат операции «XOR» равен «1» только в том случае, если «A» или «B», но не оба сразу, содержит «1».

Иными словами, результат такой же, как для операции «OR» во всех случаях, за исключением того, когда и A и B содержат «1».

XOR => OR — AND

BIT A	BIT B	BIT A «XOR» BIT B
0	0	0
1	0	1
0	1	1
1	1	0

BIT — бит; BIT A «XOR» BIT B — результат выполнения «XOR для битов A и B.

Последнее, что нам необходимо рассмотреть, это результат воздействия этих операций на флаги.

Флаг нуля. Этот флаг будет установлен (=1), если результат равен нулю;

Флаг знака. Этот флаг будет установлен (=1), если бит 7 результата установлен.

Флаг переноса. Флаг будет сброшен (=0), после «AND», «OR», «XOR», т.е. перенос будет сброшен.

Флаг четности. Этот флаг будет установлен (=1), если в результате будет четное количество единичных битов:

01101110 => установлен
01101010 => сброшен

Обратите внимание, что этот флаг также дублирует флаг переполнения.

Флаг половинного переноса, флаг вычитания. Эти флаги полезны, если используется арифметика для двоично-кодированных десятичных чисел. Оба флага сброшены (=0) после «AND», «OR», «XOR»

Применение булевских операций над флагами

Есть особый случай булевских операторов, очень удобный. Случай, когда регистр A действует на самого себя.

AND A	«A» не меняется, флаг переноса сбрасывается;
OR A	«A» не меняется, флаг переноса сбрасывается;
XOR A	«A» устанавливается равным нулю, флаг переноса сбрасывается.

Эти команды широко используются, поскольку в них нужен всего один байт для выполнения того, что в противном случае потребовало бы двух. Например:

```
LD A,0
```

Флаг переноса часто приходится сбрасывать, например, в рабочем порядке перед применением арифметических операций, таких как

ADC	сложение с переносом
SBC	вычитание с переносом

и это можно легко сделать с помощью команды AND без изменения какого бы то ни было регистра.

Выводы

Есть три логических оператора, полезных в машинном языке:

AND
OR
XOR

Эти команды действуют только на 8-битовые числа, причем одно из этих чисел должно храниться в регистре «А».

Обратите внимание, что смысл операции «AND» в машинном языке отличается от его смысла в качестве команды BASIC.

Логические операторы проверяют отдельные биты двух чисел и поэтому полезны при маскировании чисел и установке отдельных битов.

Работа с числами для двух рук

До сих пор мы работали только с числами для одной руки (8-битовыми), но мы говорили о том, что ЦП может также в некоторых случаях обрабатывать и числа для двух рук (16-битовые).

Один из этих случаев мы уже упоминали — это индексные регистры. На этих «ногах» по 16 «пальцев» (16 битов) и могут обрабатывать только 16-битовые числа.

Кроме того, мы знаем, что применяя совместно две руки, мы можем иногда запоминать 16-битовые числа. Эти совместно используемые руки мы называем «парами регистров». Это пары BC, DE и HL.

ЦП обрабатывает 16-битовые числа во многом аналогично тому, как вы или я работаем с тяжелыми предметами: нам нужны две руки, мы не очень хорошо приспособлены для работы с ними и способ работы с ними у нас медленный и с ограничениями.

Теперь давайте рассмотрим различные способы адресации (возможные видоизменения?), имеющиеся для обработки 16-битовых чисел.

Непосредственная расширенная адресация:

```
LD RR,NN
```

(или другие команды)

Это — эквивалент непосредственной адресации для 8-битовых чисел. Просто непосредственная адресация расширена таким образом, чтобы применяться для передачи 16-битовых данных.

В общем случае команды, обрабатывающие 16-битовые числа, длиннее и медленнее, чем предназначенные для 8-битовых. Например, если 8-битовые команды с непосредственной адресацией имеют длину 2 байта (один для команды и один для числа), то расширенная версия (т.е. для 16 битов) требует трех байтов.

Формат для расширенной непосредственной адресации имеет вид:

Байт 1	команда
Байт 2	N1 младший байт числа
Байт 3	N2 старший байт числа.

Мы используем этот тип адресации команды для определения содержимого пары регистров, например, в качестве указателя на ячейку памяти.

Регистровая адресация

Вы, наверно, помните, что регистровой адресацией мы называем такую адресацию команды, при которой обрабатываемое значение хранится в одном из регистров.

То же самое остается верным для 16-битовых команд, за исключением того, что в наборе команд ЦП таких команд немного. Они в основном относятся к арифметическим операциям и очень ограничены в плане допустимых комбинаций регистров. Например:

ADD HL,BC.

Мы здесь вновь отметим предпочтение, оказываемое ЦП паре регистров HL. Именно здесь проходит «мускул», и некоторые команды можно выполнить только с помощью этой пары регистров. Это верно для арифметических команд, подробнее мы на этом остановимся в одной из следующих глав.

Косвенная регистровая адресация

Косвенной регистровой адресацией мы называем такую адресацию команды, при которой требуемое значение находится в памяти, а адрес ячейки памяти хранится в паре регистров.

В Z80 этот тип адресации, опять-таки, в основном используется с помощью пары регистров HL. Например:

JP (HL).

Расширенная адресация

Расширенная адресация по смыслу близка расширенной косвенной регистровой адресации, за исключением того, что нужное значение хранится не в паре регистров, а в паре ячеек памяти.

Например, LD HL,(NN), где NN должно быть задано на стадии программирования.

Упражнение

С помощью редактора машинного языка в коде EZ, введите следующие программы:

1. Непосредственная расширенная адресация:

```
010F00 LD BC,15        : загрузить значение 15 в BC
C9 RET                 : возврат
```

Когда выполняется эта программа, вы обратите внимание на то, что значение USR после возврата из программы на машинном языке равно 15, в точности как мы задали.

Обратите внимание, насколько ограничен этот тип адресации: вы должны задать значение числа в программе.

2. Регистровая адресация

Теперь мы добавим к приведенной выше программе одну строчку:

```
210040      LD HL,4000H    : загрузить 16384 в HL
010F00      LD BC,15       : загрузить 15 в BC
09          ADD HL,BC      : сложить эти два числа
C9          RET            : возврат
```

Если вы выполните эту программу, у вас все равно получится тот же ответ, что и раньше, а именно 15! Почему? Разве мы не прибавили 16384? Ответ таков: Да, прибавили, но все это произошло в паре регистров HL, так что мы ничего этого не увидели! Чтобы увидеть, что же произошло нам нужно добавить несколько строк, следующим образом:

3. Расширенная адресация:

```
210040      LD HL,4000H
010F00      LD BC,15
09          ADD HL,BC
22647D      LD (7D64H),HL  : Поместить HL в 32100 и 32101
ED4B647D    LD  BC,(7D64H) : взять значение для BC из 32100 и 32101
C9          RET
```

Этот способ передачи информации из HL в BC на самом деле не применяется в программировании, поскольку команды PUSH и POP более эффективны, но он показывает, что приходится иногда делать, чтобы преодолеть ограниченный характер способов адресации ЦП Z80.

Вы можете просмотреть ячейки памяти 32100 и 32101 с помощью команды «MEM», чтобы проверить также и эту программу.

Обработка чисел на двух руках

В предшествующих главах мы видели, насколько проворным может быть ЦП при обработке чисел на одной руке, и мы только что рассмотрели способ, которым он может обрабатывать числа для двух рук.

Математические способности ЦП таковы, что он может выполнять очень сложные вычисления над большими числами с помощью всего одной руки. Зачем же тогда беспокоиться о числах для двух рук?

Вам встретятся случаи, когда вы обнаружите, что невозможно задать все на свете с помощью одних только 8-битовых чисел. Если бы мы были ограничены одним только диапазоном от 0 до 255 8-битовых чисел, то наша ЭВМ была бы конечно очень ограниченной.

Наиболее ярким примером необходимости 16-битовых чисел служит задание адреса ячейки памяти. Мы предполагали, что такие действия возможны, когда да рассматривали такие команды, как LD A,(HL).

Медленный способ действия состоял бы в том, чтобы загружать каждый отдельный регистр из пары регистров, как мы делали в предшествующих упражнениях.

К счастью для нас, имеются (хоть их не много) команды чипа Z80, позволяющие нам обрабатывать 16-битовые числа. В этой главе мы будем иметь дело с загрузкой 16-битовых чисел, а в следующей главе мы рассмотрим арифметические действия над 16-битовыми числами.

Задание адресов с помощью 16-битовых чисел

Обратите, пожалуйста, внимание, что все адреса задаются с помощью 16-битовых чисел.

Вы просто не сможете задать адрес с помощью всего 8 битов, даже если этот адрес лежит в диапазоне от 0 до 255. В соответствии со способом работы ЦП это не будет адресом, поскольку в нем нет двух байтов по 8 битов каждый.

Мы подразумевали это, когда использовали сокращенную запись:

```
LD A,(NN)
```

Так что помните также, что 16-битовые числа хранятся в паре регистров так, что первым идет старший байт (посмотрите еще раз нашу главу «заглянем в ЦП» (A LOOK INTO THE CPU) — здесь «HL» означает: H= «HIGH» (старший); L = «LOW» (младший)).

Хранение 16-битовых чисел в памяти

Есть один аспект конструкции Z80, который очень трудно объяснить или оправдать: при загрузке 16-битовых чисел в память используется обратный порядок по сравнению с парами регистров. Младший бит в памяти всегда хранится первым!

Давайте рассмотрим ситуацию, в которой мы помещаем в память содержимое HL:
до того:

ячейка	содержимое
32000	00

H	L			
01	02		32001	00
			32002	00

Предположим, что в HL содержится десятичное число 258 = 0102H. Все ячейки памяти пусты.

после:

		ячейка	содержимое
		32000	02
H	L	32001	01
01	02	32002	00

Порядок хранения 16-битовых чисел в памяти (и в распечатке программы) таков, что младший бит всегда хранится в начале.

Команды операций загрузки для двух рук

MNEMONIC	BYTES	TIME TAKEN	EFFECT ON FLAGS					
			C	Z	PV	S	N	H
LD REG PAIR,NUMBER	3/4	10	-	-	-	-	-	-
LD IX,NUMBER	4	14	-	-	-	-	-	-
LD IY,NUMBER	4	14	-	-	-	-	-	-
LD (ADDRESS),BS OR DE	4	20	-	-	-	-	-	-
LD (ADDRESS),HL	3	16	-	-	-	-	-	-
LD (ADDRESS),IX	4	20	-	-	-	-	-	-
LD (ADDRESS),IY	4	20	-	-	-	-	-	-
LD BC OR DE,(ADDRESS)	4	20	-	-	-	-	-	-
LD HL,(ADDRESS)	3	16	-	-	-	-	-	-
LD IX,(ADDRESS)	4	20	-	-	-	-	-	-
LD IY,(ADDRESS)	4	20	-	-	-	-	-	-

MNEMONIC — мнемоническое обозначение; BYTES — количество байтов; TIME TAKEN — время выполнения; EFFECT ON FLAGS — воздействие на флаги; REG PAIR — пара регистров; NUMBER — число; ADDRESS — адрес; OR — или.

Обозначения для флагов:

\# означает, что флаг изменился в результате операции;
0 означает, что флаг сбрасывается;
1 означает, что флаг устанавливается;
– означает, что флаг не изменяется.

Такому решению нет никакого оправдания, можно только сказать, что так решили конструкторы Z80 и нам теперь приходит с этим мириться.

Очень желательно, чтобы вы внимательно это прочли и удостоверились, что освоили это обращение традиционных правил. Скорее всего именно это станет единственным важнейшим источником ошибок в программах.

В регистрах: старший бит хранится первым;

В памяти и программах: младший бит хранится первым.

Этот факт нельзя прочитать и забыть, поскольку каждый раз, когда вам придется иметь дело с 16-битовой командой на машинном языке нужно будет тщательно продумать порядок младших и старших битов.

Однако не стоит из-за этого расстраиваться — работать на Z80 без 16-битовых команд практически невозможно, так что это цена, которую приходится платить.

Вы сами можете проверить этот факт, «выполнив» эту команду с помощью «редактора машинного языка в коде EZ» и затем прочитав содержимое памяти с помощью команды «MEM».

Загрузка 16-битовых чисел

Группа команд загрузки 16-битовых чисел, говоря упрощенно, сводится к загрузке 16-битового числа в пару регистров. Общий вид мнемонического сокращения следующий

```
LD RR, NN
```

Вновь мы применяем двухбуквенное обозначение для 16-битовых чисел. «RR» обозначает произвольную пару регистров, «NN» произвольное 16-битовое число.

Для тех из вас, кто лишен преимуществ ассемблера, т.е. если вам приходится вручную переводить мнемонические обозначения в машинный код с помощью помещенных в конце книги таблиц, то рассмотренный нами вопрос о порядке 16-битовых чисел в памяти становится принципиальным.

Даже если у вас есть ассемблер, нужно знать об этом обратном порядке, чтобы уметь «читать» текст машинной программы при просмотре памяти.

Давайте рассмотрим конкретный пример:

Загрузить 258 в HL мнемоническое обозначение будет

```
LD HL,0102H
```

Команда, соответствующая LD HL,NN, как можно удостовериться, посмотрев в конец книги, будет:

```
21 XX XX
```

Это означает, что вместо «xx xx» нужно вставить число 0102H. Но из-за правила перестановки мы не станем вводить его в виде 1002H (так в оригинале. Очевидно, опечатка. Должно быть 0102H. Примеч. пер.).

Поэтому правильно будет записать эту команду так:

```
21 02 01
```

В наших примерах мы будем показывать это так:

```
21 02 01 LD HL,0102H (= 258)
```

У вас может не возникнуть трудностей при вводе наших программ, но нужно очень ясно себе это представлять, чтобы трудности не возникли при написании вами собственных программ.

Другие команды загрузки для 16 битов

Помимо возможности загрузки 16-битовых чисел непосредственно в пары регистров у нас есть еще возможность загрузки 16-битовых чисел непосредственно в индексные регистры (как вы помните, оба они представляют собой ноги с 16 пальцами).

```
LD IX,NN
LD IY,NN
```

Мы можем также управлять обменом информацией между парой регистров и двумя последовательными ячейками памяти. (это 16-битовый эквивалент загрузки информации из одного регистра в одну ячейку памяти).

Общий вид команды следующий

```
LD (NN),RR
LD (NN),IX
LD (NN),IY
```

Напомним, что скобки — это сокращенное обозначение «содержимого», так что последняя команда читается как «загрузить содержимое регистра IY в ячейку памяти NN».

Поскольку мы имеем дело с 16-битовыми числами, на самом деле мы загружаем указанную ячейку памяти и следующую за ней в пару регистров. Нет необходимости задавать оба адреса (поскольку ЦП может вычислить адрес второй ячейки), но надо проявить внимание, чтобы не спутать 8-битовые операции с 16-битовыми.

Здесь также проявляется взаимообратный характер многих команд, так что мы можем также загрузить в пару регистров или в индекеный регистр все, что ни находится в конкретной паре ячеек памяти:

```
LD RR,(NN)
LD IX,(NN)
LD IY,(NN)
```

Упражнение

Мы знаем из руководства «SPECTRUM», что получить начало свободного пространства памяти можно, проверив содержимое ячеек памяти 23653 и 23654.

На языке BASIC мы можем задать это с помощью строки:

```
PRINT PEEK 23653 + 256 * PEEK 23654
```

Мы теперь выполним то же самое действие с помощью машинного языка:

```
(23653 = 5C65H)
ED 4B 65 5C    LD BC,(23653)
C9             RET
```

Обратите внимание на то, что числа вводятся так, что первым идет младший байт, и у вас совершенно неверный ответ, если будете вводить их в обратном порядке.

При работе на «SPECTRUM» нам известно, что, как только программа закончена, положение свободной области памяти оказывается фиксированным и нам его нужно определить только один раз для каждой программы.

Мы используем регистр BC для получения информации, поскольку, как вы помните, значением USR является содержимое пары регистров BC при завершении программы на машинном языке.

Обратите внимание, что команда «LD BC,(NN)» состоит из четырех байтов!

Вы можете с помощью аналогичных программ определять значение произвольных двухбайтовых переменных, перечисленных в руководстве «SPECTRUM».

Выводы

Мы можем загрузить 16-битовые числа в любую из пар регистров или в индексные регистры либо задавая 16-битовое число, либо ячейку памяти, где оно должно находиться.

Аналогичным образом мы можем передавать в память 16-битовое слово из любой из пар регистров или из индексных регистров.

Единственное, на что нужно обратить особое внимание — это специфический порядок, в котором хранятся 16-битовые числа центральным процессором Z80 в памяти (и тем самым в командах программы, использующих 16-битовые числа):

Младший байт всегда хранится первым!!

Работа со стеком

Вы можете вспомнить развитое нами в начале книги представление о стеке как месте, в котором ЦП может хранить информацию, не заботясь о запоминании ее конкретного адреса.

Команды операций со стеком

MNEMONIC	BYTES	TIME TAKEN	EFFECT ON FLAGS					
			C	Z	PV	S	N	H
PUSH REG PAIR	1	11	-	-	-	-	-	-
PUSH IX OR IY	2	15	-	-	-	-	-	-
POP REG PATR	1	10	-	-	-	-	-	-
POP IX OR IY	2	14	-	-	-	-	-	-
LD SP,ADDRESS	3	10	-	-	-	-	-	-
LD SP,(ADDRESS)	3	20	-	-	-	-	-	-
LD SP,HL	1	6	-	-	-	-	-	-
LD SP IX OR IY	2	10	-	-	-	-	-	-

MNEMONIC — мнемоническое обозначение; BYTES — байты; TIME TAKEN — время выполнения; EFFEST ON FLAGS — результат воздействия на флаги; REG PAIR — пара регистров; OR — или; ADDRESS — адрес.

Обозначения для флагов:

- # означает, что флаг изменяется в результате операции;
- 0 означает, что флаг сбрасывается;
- 1 означает, что флаг устанавливается;
- – означает, что флаг не изменяется в результате операции.

Одно из преимуществ, возможно не очень существенных, операций со стеком состоит в том, что мы можем вталкивать и выталкивать информацию порциями по 2 руки (16 битов). Это происходит потому, что стек прежде всего сконструирован для запоминания адресов, а нам необходимо задавать адреса в виде 16-битовых чисел.

Общий вид команд вталкивания информации в стек следующий:

```
PUSH RR
PUSH IX
PUSH IY
```

А общий вид команд выталкивания информации назад из стека:

```
POP RR
POP IX
POP IY
```

Это — очень простые команды, и вы, конечно, заметите, что нет необходимости задавать адрес.

Для обычных пар регистров (т.е. не индексных регистров) эти команды имеют длину всего один байт и поэтому экономны с точки зрения практики программирования.

Команды PUSH, кроме того, не портят регистров, т.е. 16-битовый регистр после выполнения команды PUSH продолжает содержать ту же информацию, что и до нее.

Обратите внимание, что поскольку мы можем выполнять операции PUSH и POP для произвольных пар регистров, регистр, для которого выполняется команда POP не обязан быть тем же самым, для которого выполнялась команда PUSH! Например:

```
PUSH BC
POP HL
```

Результат выполнения этих двух команд состоит в том, что содержимое регистра (так в оригинале, точнее было бы говорить о паре регистров. Примеч. пер.) BC не изменяется, а содержимое HL становится равным содержимому регистра BC во время выполнения команды PUSH.

Это по существу добавляет команду типа:

```
LD RR,RR'
```

к группе команд загрузки 16-битовых чисел, которая явно отсутствовала.

Поскольку каждая из команд PUSH и POP для пар регистров имеет длину всего 1 байт, затраты с точки зрения памяти не очень велики.

Другим дополнительным преимуществом является то, что мы можем выполнять команды PUSH и POP для пары регистров AF! Это — одна из немногих команд, в которых AF рассматривается как пара регистров, и очевидно в этом есть смысл, поскольку нам не раз придется сохранять содержимое флагов.

Это означает, что вы можете выполнить PUSH AF (по существу записать, чему равны A и F), провести вычисления, нежелательным побочным эффектом которых может быть изменение флагов, и затем выполнить POP для AF, оставляя флаги неизменными.

Работа со стеком

Как вы знаете, действительные преимущества команд PUSH и POP состоят в том, что нам не приходится думать об адресах чисел, над которыми эти команды выполняются.

Вы, безусловно, согласитесь, что не всегда имеет смысл, чтобы одна и та же область памяти служила стеком независимо от того, будет у вас 16K или 48K.

На самом деле ЦП отслеживает адрес стека с помощью «указателя стека», который можно считать 16-битовым регистром. Мы кратко упомянули об этом, когда рассматривали регистры, но ничего не говорили об этом при рассмотрении команд LOAD и тому подобных, поскольку это не такой регистр, который можно обрабатывать так же, как остальные регистры.

Основное, что желательно уметь делать с указателем стека, это определять его положение в памяти, и именно такой тип команды имеется:

```
LD SP,NN
LD SP,(NN)
LD SP,IX
LD SP,IY
```

Вы можете проверить стек «SPECTRUM» с помощью команды «MEM» программы «редактора машинного языка в коде EZ», просматривая последние 30 — 40 байтов перед RAMTOP. Не изменяйте содержимое ячеек стека.

Почти любое изменение приведет к остановке «SPECTRUM», э ран погаснет и вам придется вновь включать питание. Так происходит потому, что операционная система помещает много информации, необходимой ей, в стек, и изменения приводят к тому, что она идет вразнос.

По этой же самой причине не пытайтесь менять положение указателя стека, если только у вас нет полной уверенности в том, что вы делаете.

Замечание:

В правильно организованной программе количество команд POP и PUSH должно совпадать, независимо от того, по какой ветви шла программа. Любые ошибки в подсчете могут привести к странным результатам.

Упражнение:

Мы можем применить эти команды для проверки адреса, по которому вызывается программа USR путем выталкивания значения из стека в регистр возврата BC. Как это делается, показывает следующая программа:

```
C1 POP BC    : GET ADDRESS IN BC
C5 PUSH BC   : PUT IT BACK ON THE STACK
C9 RET
```

GET ADDRESS IN BC — получить адрес в BC; PUT IN BACK ON THE STACK — поместить его обратно в стек.

Арифметические команды для двух рук

MNEMONIC	BYTES	TIME TAKEN	EFFECT ON FLAGS					
			C	Z	PV	S	N	H
PUSH REG PAIR	1	11	-	-	-	-	-	-
ADD HL,REG PAIR	1	11	#	-	-	-	0	?
ADD HL,SP	2	11	#	-	-	-	0	?
ADC HL,REG PAIR	2	15	#	#	#	#	0	?
ADC IX,SP	2	15	#	#	#	#	0	?
ADD IX,BC OR DE	2	15	#	-	-	-	0	?
ADD IX,IX	2	15	#	-	-	-	0	?
ADD IX,SP	2	15	#	-	-	-	0	?
ADD IY,BC OR DE	2	15	#	-	-	-	0	?
ADD IY,IY	2	15	#	-	-	-	0	?
ADD IY,SP	2	15	#	-	-	-	0	?
SBC HL,REG PAIR	2	15	#	#	#	#	1	?
SBC HL,SP	2	15	#	#	#	#	1	?

MNEMONIC — мнемоническое обозначение; BYTES — байты; TIME TAKEN — время выполнения; EFFECT ON FLAGS — воздействие на флаги; REG PAIR — пара регистров; OR — или;

Обозначение флагов

\# означает, что флаг изменяется в результате операции;

0 означает, что флаг сбрасывается;

1 означает, что флаг устанавливается
– означает, что флаг не изменяется;
? означает, что результат неизвестен.

Арифметические действия для двух рук

Одно из преимуществ возможности 16-битового представления чисел в, по сути дела, 8-битовом процессоре состоит в том, что мы можем применять 16 битов для задания адреса и для выполнения вычислений над целыми числами до 65 355 (или в диапазоне от -32768 до 32767, если допускаются отрицательные числа).

В свете вышесказанного легко понять, почему в некоторых ранних моделях микро-ЭВМ, таких как первоначальная модель «СИНКЛЕР-ZX80», все арифметические действия в языке BASIC ограничивались целыми числами из диапазона -32 000 до +32 000.

Но даже несмотря на то, что мы можем выполнять некоторые арифметические действия с помощью двух рук, наш заголовок этой главы содержит намек на то, что будет дальше — арифметические действия над числами для двух рук несколько менее удобны, чем для одной руки. Не хватает широкого спектра вариантов!

Привилегированная пара регистров

Совершенно так же, как регистр «А» является привилегированным с точки зрения арифметических действий для одной руки, есть и привилегированная пара регистров для арифметических действий над числами для двух рук, и это — пара регистров HL.

Эта привилегированность не столь явно выражена, как в случае 8-битовых чисел, так что мы не опускаем имя пары регистров.

Сложение

Операции сложения вполне прямолинейны:

```
ADD  HL, BC
ADD  HL, DE
ADD  HL, HL
ADD  HL, SP
```

Но именно так они и записываются!

Обратите внимание, что не удается сложить абсолютное число с HL — например не допускается «ADD HL,NN». Чтобы выполнить вычисления такого типа, нам нужно:

```
LD  DE,NN
ADD  HL,DE
```

Когда вы учтете, что теперь у вас связанными оказались четыре из 8-битовых регистра, которых всего 7, вы сразу поймете, что вы не захотите делать это слишком часто.

Обратите также внимание, что между HL и индексными регистрами нельзя выполнить сложение. Вы также вспомните, что нет команды LOAD, которая позволяла бы вам передавать содержимое IX или IY в BC или DE, так что единственным способом выполнения такого сложения было бы:

```
PUSH  IX
POP  DE
ADD  HL,DE
```

Один из моментов, о котором следует упомянуть — это регистр «SP» — указатель стека. Это — одна из очень немногих операций, в которых «SP» обрабатывается как настоящий регистр, но, очевидно, вы не можете применять его в качестве переменной! Подумайте, что

случилось бы со всеми командами POP и PUSH, если бы вы произвольным образом меняли содержимое указателя стека!

Воздействие на флаги

16-битовые арифметические операции — это именно та область, в которой флаг переноса играет присущую ему роль, поскольку, как можно видеть из таблицы, помещенной в начале этой главы, единственный другой флаг, на который оказывает влияние команда «ADD» (сложение) — это флаг «вычитания» (и все что он в этом случае означает — это то, что команда сложения не является вычитанием).

Флаг переноса будет установлен, если имеется переполнение в старшем бите «H» — любое переполнение в «L» автоматически помещается в результате вычисления в «H».

Сложение с переносом

Из-за ограниченного характера работы с 16-битовыми числами мы можем выполнять сложение цепочкой точно так же, как и в 8-битовом случае. Команда «сложение с переносом» — ее мнемоническое обозначение «ADC» (ADD WITH CARRY) — действует аналогично команде «ADD» и имеет тот же диапазон пар регистров:

```
ADC HL,BC
ADC HL,DE
ADC HL,HL
ADC HL,SP
```

16-битовое вычитание

16-битовое вычитание — тоже достаточно простая операция, но вычитания без переноса не бывает:если вы не уверены в состоянии флага переноса, удостоверьтесь, что в вашей программе имеется строка, очищающая флаг переноса перед каждой операцией вычитания.

```
SBC HL,BC
SBC HL,DE
SBC HL,HL
SBC HL,SP
```

У последней команды имеется очевидное применение: установить HL на конец памяти, используемой вашей программой, экраном дисплея и переменными, вычесть SP, и результат (отрицательный) будет задавать количество свободной памяти. Сможете ли вы написать простую программу, выполняющую все это? Посмотрев в конец главы, вы сможете проверить свое решение.

Воздействие арифметических операций с переносом на флаги

Вы, возможно, заметили, что три других флага изменяются в результате команд «сложение с переносом» и «вычитание с переносом», хотя они не изменялись в результате простых команд 16-битового сложения. Эти флаги — флаг нуля, флаг знака и флаг переполнения. Каждый из них устанавливается в соответствии с результатом операции.

Арифметические операции с индексным регистром

Индексные регистры ограничиваются только операцией сложения без переноса.

Более того, диапазон регистров, которые можно складывать с индексными регистрами, чрезвычайно ограничен:

Сложение с парой регистров «BC» или «DE»; Сложение индексного регистра с самим собой; Сложение с указателем стека.

Упражнение на решение задачи о свободной памяти

Конец пространства памяти, используемого программой, определяется содержимым ячейки памяти STKEND. В руководстве по «SPECTRUM» она определяется адресами 23653 и 23564.

Очевидно, если мы загрузим содержимое этой ячейки в HL, то задача наполовину будет решена:

```
LD HL,(STKEND)
```

Затем нужно вычесть «указатель стека» (SBC HL,SP?).

Из-за «переноса» нам необходимо очистить флаг переноса. Наиболее легко это достигается с помощью команды «AND A», рассмотренной в этой книге раньше.

```
AND A
SBC HL,SP
```

Можете считать задачу решенной на три четверти, если вы знали, что нужно учесть перенос, но не знали, как это сделать. Если вы вовсе забыли о переносе, задача решена на четверть.

Поскольку, указатель стека задает положение в памяти выше, чем наибольший адрес вашей программы (иначе у вас возникли бы серьезные трудности), результат будет отрицательным.

Теперь давайте получим количество байтов, оставшихся свободными, в виде положительного числа с помощью регистра «BC» (точно так же для этого подошел бы и регистр «DE»). Сначала мы хотим сдвинуть HL и BC, но нет команды «загрузить». Для такого действия нам придется использовать последовательно команды PUSH и POP:

```
PUSH HL
POP BC
```

В HL по-прежнему находится та же информация, так что HL=BC.

Чтобы получить HL = -BC, нужно вычесть BC из HL дважды (но не забудьте, что флаг переноса только что был установлен командой вычитания, так что его необходимо вновь очистить):

```
AND A
SBC HL,BC
SBC HL,BC
```

В HL теперь содержится отрицательное значение, по модулю равное прежнему т.е. положительное число оставшихся свободными байтов.

Теперь нам нужно получить это число снова в паре регистров BC, чтобы получить результат из функции «USR». Чтобы получить HL назад в BC:

```
PUSH HL
POP BC
```

И, наконец, возврат из функции USR:

```
RET
```

Правильно ли вы написали программу? Обратили ли вы внимание, как удобно пользоваться стеком!

Циклы и переходы

Именно циклы и переходы дают большие возможности программированию. Получая возможность принимать решения и выполнять различные участки программы в

зависимости от предшествующих вычислений, вы действительно приобретаете новые возможности.

Это свобода может также служить источником трудностей, приводить к программам, трудным для понимания и почти недоступным для отладки.

Я настоятельно рекомендую сначала тщательно спроектировать свою программу, прежде чем писать что-либо на машинном языке, и именно поэтому мы включили главу «планирование вашей программы на машинном языке». Я подчеркиваю это теперь потому, что циклы и переходы — это то, что соблазняет отклониться от правильного проектирования программы.

Эквивалент «GO TO» на машинном языке

В «Бейсике» вы знаете команду «GO TO», передающую управление вашей программой командам в той строчке, на которую указывает «GO TO».

Нет ничего проще, чем реализовать это на машинном языке, просто задайте ячейку памяти, в которой вы хотели бы, чтобы ЦП обнаружил следующую команду, и задача наполовину решена. Наиболее простой вариант — команда «перейти на» (JUMP TO)

```
JP XX XX
JP (HL)
JP (IX)
JP (IY)
```

Можно сделать так, чтобы одна из этих команд зависела от состояния одного из флагов, напр[имер] флага переноса. Это команда условного перехода имеет вид:

```
JP CC,NN
```

Где CC — условие, выполнение которого проверяется. Если бы у нас было, например,

```
JP Z, 0000
```

То это читалось бы «переход по адресу 0000, если флаг нуля установлен». (это — адрес, по которому «SPECTRUM» осуществляет переход при включении питания, и в таком виде команда «JP» на ноль может применяться в программе на машинном языке, если вы захотели очистить всю память и начать заново с помощью «К»).

Теперь обратите внимание, что ЦП не допускает никаких ошибок. Если вы говорите «JUMP», он сделает переход. Поскольку почти любой код может быть воспринят как команда, ЦП не принимает во внимание, что вы могли сделать переход в середину массива данных или во второй байт двухбайтовой команды: он будет считывать байт по найденному адресу и считать, что это — начало следующей команды.

Способ, которым ЦП обрабатывает команду перехода на самом деле совершенно прост: у него есть небольшой счетчик, называемый счетчиком команд, говорящий ему, где искать следующую выполняемую команду. При нормальном ходе программирования (т.е. без переходов) ЦП проверяет подлежащую выполнению команду и добавляет к счетчику команд столько, сколько байтов содержит команда.

Так, если он встречает 2-байтовую команду, он добавляет 2, тогда как 4-байтовая команда заставит его добавить 4 к счетчику команд.

Когда он встречает команду «перехода», он просто замещает содержимое счетчика команд тем значением, которое вы указали. Именно поэтому вы не можете допустить проникновения в программу каких-либо ошибок.

Длинные переходы и короткие переходы

Мы можем считать приведенные выше команды эквивалентом длинного перехода в машинном языке, поскольку 16-битовый адрес позволяет нам осуществить переход в любое место, куда позволяет перейти чип Z80.

Недостатки длинного перехода:

а) Часто нам нет необходимости в таком длинном переходе, но нам все-таки приходится применять 3-байтовую команду.

б) Мы не можем без затруднений переместить программу в другую часть памяти, потому что мы задаем абсолютный адрес.

Именно для преодоления этих двух недостатков был введен «короткий переход». Он называется «относительным переходом» и позволяет нам совершать переход от текущей позиции на +127 или -128 байтов, т.е. расстояние перехода можно задать в одном байте!

Команда относительного перехода

```
JP Д,
```

Где Д — относительное смещение.

Мы можем также сделать относительный переход в зависимости от некоторого условия, например, от того, установлен ли флаг переноса или флаг нуля. Эти условные переходы записываются следующим образом:

```
JP СС,Д,
```

Где СС — проверяемое условие.

Значение смещения Д добавляется к счетчику команд.

Это означает, что берется текущее значение счетчика команд и к нему прибавляется заданное вами относительное значение. Вы можете задавать либо положительное (переход вперед), либо отрицательное (переход назад) значение. Если вы посмотрите, в предшествующей главе об отрицательных числах, то вы поймете, что это означает ограничение относительных переходов, диапазоном от -128 до +127.

Обратите внимание, что когда ЦП выполняет команду относительного перехода, счетчик команд уже указывает на следующую команду, которая выполнялась бы, если условие не удовлетворялось.

Так происходит потому, что когда ЦП встречает «JP», он знает что имеет дело с 2-байтовой командой, и добавляет 2 к счетчику команд — поэтому счетчик команд указывает на команду, следующую за относительным переходом!

Пример. В такой программе, как

```
Ячейка        Текст
32000         ADD A,B
32001         JP Z,02H
32003         LD B,0
32005  NEXT   LD HL,4000H
```

Ниже показано, как ЦП работает с этой программой, если игнорируется команда перехода, расположенная по адресу 32001 (т.е. при сброшенном флаге нуля):

Загрузить байт по адресу 32000 поскольку в этом байте содержится команда, состоящая только из одного байта, счетчик команд нужно задать равным 32001.

Выполнить команду.

Загрузить байт, заданный счетчиком команд (32001). Этот байт будет являться частью 2-байтовой команды, так что нужно прибавить 2 к счетчику команд и сделать его равным 32003.

Получить следующий байт, чтобы сделать команду полной.

Выполнить команду.

Загрузить байт, заданный счетчиком команд (32003). Этот байт является частью 2-байтовой команды, так что нужно прибавить к счетчику команд 2 (теперь он становится равным 32005).

Получить следующий байт, чтобы сделать команду полной.

Выполнить команду. В ячейке 32001 программа встречает команду относительного перехода. Если флаг нуля не установлен, как в приведенном выше примере, то ЦП ничего не выполняет. В общем случае ЦП выполняет команду перехода следующим образом:

Если флаг нуля установлен, прибавить 2 к счетчику команд (это составит 32005).

Если флаг нуля сброшен, не нужно делать ничего (счетчик команд остается равным 32003). Иными словами, относительный переход позволяет нам в определенных случаях перескакивать через команду «LD B,0».

Это также объясняет, почему для этой команды указаны два времени выполнения. Подсчет нового значения счетчика команд занимает больше времени, чем невыполнение никаких действий.

Поэтому ЦП выполнит либо команду, расположенную по адресу 32003, либо — по адресу 32005, в зависимости от значения флага нуля.

Как мы уже говорили, можно также сделать относительный переход с отрицательным значением.

Упражнение

Поскольку относительный переход — 2-байтовая команда, и счетчик команд после относительного перехода указывает на следующую команду, к какому результату приведет такая команда:

```
JP -2 ?
```

Циклы «FOR.... NEXT» на машинном языке

Вы, я уверен, знакомы с формой цикла «FOR... NEXT» языка BASIC:

```
FOR I = 1 TO 6
LET C = C+1
NEXT I
```

Эквивалентное этому предложению на машинном языке сходно по смыслу, но приобретает несколько иную форму. Давайте рассмотрим как мы могли бы реализовать цикл на машинном языке с помощью арифметических функций и относительных переходов:

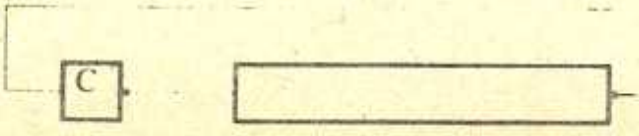
```
        LD  B,1         : SET COUNTER TO 1
        LD  A,7         : MAX. OF COUNTER + 1
LOOP    INC C           : C = C+1
        INC B           : INCREMENT COUNTER
        CP B            : IS B = A?
        JR NZ,LOOP      : IF NOT LOOP AGAIN
```

SET COUNTER TO 1 — установить счетчик равным 1; MAX. OF COUNTER — максимальное значение счетчика; INCREMENT COUNTER — увеличение счетчика; IS B=A? — равно ли; IF NOT LOOP AGAIN — если нет, повтор цикла.

Такая программа будет работать, но нужно заметить следующее.

Мы связываем пары регистров, одну для увеличения, а одну для хранения максимума; и команда наращивания счетчика по завершении не знает никаких флагов.

Гораздо лучше было бы уменьшать значение счетчика!



Мы знаем, что должны выполнить цикл 6 раз, так почему не задать «B» равным 6 и не уменьшать его?

Тогда у нас получится такая программа:

```
        LD B,6          : SET COUNTER
LOOP    INC C           : C = C+1
        DEC B           : DECREASE COUNTER
        JR NZ,LOOP      : LOOP IS NOT FIGISHED
```

SET COUNTER — установить счетчик; DECREASE COUNTER — уменьшить счетчик; LOOP IS NOT FINISHED — цикл не закончен.

Вы можете заметить, что такой способ гораздо эффективнее.

У чипа Z80 есть специальная команда, соединяющая последние две приведенные выше строки.

Эта команда записывается так

DJNZ Д

И читается «уменьшить (B) и перейти, если не ноль». (здесь Д относительное смещение). Это — двухбайтовая команда, и поэтому она дает экономию в один байт по сравнению с приведенной выше текстом.

Из-за наличия этой специальной команды регистр «B» обычно используется для реализации счетчиков.

Ограничение на команду «DJNZ» состоит в том, что можно считать только до 256. Тем не менее команды DJNZ могут быть вложенными, если есть такая необходимость:

```
        LD B,10H            : B=16
        BIGLOOP PUSH BC     : SAVE VALUE OF «B»
        LD B,0              : SET B=256
LITLOOP .....
                            : WHATEVER CALCULATION
        .....
        DJNZ LITLOOP        : DONE 256 TIMES?
        POP BC              : GET BACK VALUE OF B
        DJNZ                : DO BIGLOOP 16 TIMES
```

SAVE VALUE OF «B» — запомнить значение в «B»; SET — задать; WHATEVER CALCULATION — произвольные вычисления; DONE 256 TIMES — выполнился 256 раз?; GET BACK VALUE OF B — получить вновь значение B; DO BIGLOOP 16 TIMES — выполнить охватывающий цикл 16 раз.

Упражнение

Запишите на кусочке бумаги, что будет находиться в каждом регистре после выполнения каждой команды в приведенной выше программе.

Циклы ожидания

В программах на машинном языке бывают ситуации, когда события происходят с такой большой скоростью, что необходимо ввести некоторое ожидание.

Примеры, сразу приходящие на ум — посылка информации на кассету магнитофона (сигналы должны быть расположены достаточно далеко друг от друга, чтобы их потом можно было прочитать) или посылка информации на печатающую машинку (представьте себе только печать со скоростью нескольких тысяч знаков в секунду).

Поэтому полезно установить циклы ожидания с помощью команды DJNZ:

```
LD B,COUNT
WAIT DJNZ WAIT
```

COUNT — счетчик; WAIT — ожидание

Команда «DJNZ WAIT» заставит ЦП возвратиться к команде DJNZ столько раз, сколько нужно, чтобы регистр «B» вновь обнулился, прежде чем обработка пойдет дальше.

Это должно вам дать ответ на наше упражнение, в котором спрашивалось, что произойдет, если вы напишете:

```
WAIT JP WAIT
```

Вам придется довольно долго ждать, пока ЦП выйдет из этого цикла!

Команды группы вызова и возврата

MNEMONIC	BYTES	TIME TAKEN	EFFECT ON FLAGS					
			C	Z	PV	S	N	H
CALL ADDRESS	3	17	-	-	-	-	-	-
CALL CC,ADDRESS	3	10/17	-	-	-	-	-	-
RET	1	10	-	-	-	-	-	-
RET CC	1	5/11	-	-	-	-	-	-

MNEMONIC — мнемоническое обозначение; BYTES — количество байтов; TIME TAKEN — время выполнения; EFFEST ON FLAGS — воздействие на флаги; ADDRESS — адрес.

Замечание: CC — условие, которое должно выполняться для исполнения команды. Ниже приводятся используемые условия:

Флаг	Сокращение	Значение
Перенос	C	установка флага переноса (=1)
	NC	сброс флага переноса (=0)
Ноль	Z	установка флага нуля (=1)
	NZ	сброс флага нуля (=0)
Четность	PE	флаг четности четный (=1)
	PO	флаг четности нечетный (=0)
Знак	M	знак минус (=1)
	P	знак положительный (=0)

Воздействие на флаги: обратите внимание, что ни один из флагов не изменяется при командах вызова и возврата. Время выполнения: если указаны два значения времени, то более короткое относится к случаю, когда условие не выполняется.

Применение подпрограмм в ваших программах на машинном языке

Применение подпрограмм в машинном языке столь же просто, как в обычных программах на языке BASIC, если не проще.

На самом деле, вы помните, что применяя функцию «USR» в своей программе на языке BASIC, вы в действительности вызываете подпрограмму, и, как вы помните, для завершения нам нужна команда «RETURN»!

Поэтому вам очень просто проверить определенные подпрограммы независимо от вашей основной программы на машинном языке.

Основное различие, с которым вам придется столкнуться при реализации подпрограмм в ваших программах на машинном языке, состоит в том, что вам нужно знать адрес начала подпрограммы.

Это может вызвать трудности, если вы храните программы на машинном языке в переменном массиве, поскольку адрес этой переменной не обязательно фиксирован. Это также означает, что программа на машинном языке не может быть легко перемещена в новую позицию памяти, если в ней используются подпрограммы.

Подпрограммы также можно вызывать в зависимости от условий. Это — эквивалент на машинном языке следующего предложения языка BASIC:

IF (CONDITION) THEN GOSUB (LINE)

CONDITION — условие; LINE — строка.

Находясь в подпрограмме, нужно проявлять осторожность, чтобы не изменить какие-либо флаги или регистры, требующиеся для дальнейших сравнений. Это необходимо для того, чтобы вы не ушли на ветвь алгоритма снова по следующему предложению «CALL» «после возврата туда, откуда вышли».

Различие состоит в том, что единственными допустимыми условиями являются состояния четырех флагов:

Флаг переноса;

Флаг нуля;

Флаг четности (а также флаг переполнения)

Флаг знака.

Напомним, что все эти флаги устанавливаются в соответствии с последней командой, повлиявшей на этот конкретный флаг.

Поэтому хороший стиль программирования предполагает помещение команды «CALL» или «RETURN» непосредственно после команды, устанавливающей флаг.

Например:

```
LD  A,(NUMBER)
CP  1
CALL Z,ONE
CP 2
CALL Z,TWO
CP 3
CALL Z,THREE
```

NUMBER — число; ONE — один; TWO — два; THREE — три.

Приведенная выше программа позволяет вам переходить к различным программам, в зависимости от значения, хранимого в ячейке «NUMBER» (число), но обратите внимание, что она предполагает, что подпрограммы не изменяют значение в регистре A!! (почему?).

Можно применить и более короткую программу, если вам известно, что для хранимого в «NUMBER» значения имеются только три заданные выше возможности:

```
LD  A,(NUMBER)
CP  2
CALL Z,TWO        : A = 2
CALL C,ONE        : A (2 => A =1
CALL THREE        : A (2 => A =3
```

NUMBER — число; TWO — два; ONE — один; THREE — три.

Так происходит потому, что команда «CP 2» устанавливает и флаг нуля, и флаг переноса, а команда вызова не влияют на какие бы то ни было флаги.

Аналогичным образом, очень полезным оказывается условный возврат из подпрограммы. (но это не считается хорошим стилем программирования).

Команды группы сравнения и перемещения блоков

MNEMONIC	BYTES	TIME TAKEN	EFFECT ON FLAGS					
			C	Z	PV	S	N	H
CALL ADDRESS	3	17	-	-	-	-	-	-
LDI	2	16	-	-	#	-	0	0
LDD	2	16	-	-	#	-	0	0
LDIR	2	21/16	-	-	0	-	0	0
LDDR	2	21/16	-	-	0	-	0	0
CPI	2	16	-	#	#	#	1	#
CPD	2	16	-	#	#	#	1	#
CPIR	2	21/16	-	#	#	#	1	#
CPDR	2	21/16	-	#	#	#	1	#

MNEMONIC — мнемоническое обозначение; BYTES — количество байтов; TIME TAKEN — время выполнения; EFFECT ON FLAGS — воздействие на флаги.

Обозначение флагов: # показывает, что флаг изменяется в результате операции; 0 показывает, что флаг сбрасывается; 1 показывает, что флаг устанавливается; — показывает, что флаг не меняется.

Время выполнения:

Для команд повторения указанное время относится к каждому циклу. Более короткое значение относится к прерыванию команды, например, для CPIR либо BC=0, либо A=(HL).

Операции над блоками

К настоящему моменту вы должны быть вполне знакомы с языком, на котором говорит ваша ЭВМ — это очень похоже на изучение иностранного языка: вы понимаете, что овладели языком, когда можете думать на нем.

В этой главе рассматривается последний набор очень полезных программ, в нескольких последующих главах мы будем иметь дело с командами, которые приятно иметь под рукой и которые при определенных обстоятельствах играют свою роль, но в общем вы должны уметь писать машинные программы на машинном языке с помощью уже известного вам материала.

Тем не менее, обязательно прочтите главу о планировании вашей программы на машинном языке!

Рассматриваемые в этой главе команды по самой своей природе способны одним махом охватывать длинные конструкции быстрее летящей пули, иными словами, команды, обрабатывающие целые блоки памяти, а не отдельные 8-битовые байты.

Давайте начнем с простейшей из них:

CPI

При вашем знании языка Z80, вы должны сразу распознать одного из представителей семейства «сравнений», и на самом деле это и есть расширенное сравнение.

По-русски она читается «сравнить и дать приращение». (вы помните, что сравнивать что бы то ни было можно только с содержимым регистра «А», и об этом в команде не требуется упоминать).

Команда «CPI» сравнивает «А» с (HL) и автоматически увеличивает HL. Это означает, что после операции CPI HL уже указывает на следующую ячейку и готов к повторению ее.

С помощью этой команды мы могли бы написать программу просмотра всей памяти в поисках конкретного совпадения следующим образом:

```
SEARCH      CPI
            JR N2,SEARCH
```

SEARCH — поиск.

При таком способе, пока не будет обнаружено совпадение (будет установлен флаг нуля, как во всех операциях сравнения) программа будет продолжать просмотр.

К сожалению, это — не такая хорошая идея, поскольку, если совпадение не будет обнаружено, программа никогда не закончится! К счастью конструкторы языка Z80 позаботились об этом, и команда CPI также автоматически уменьшает BC!

Поэтому мы можем по желанию выбирать длину блока, который мы хотим просмотреть, и таким образом задать конец просмотра.

Давайте предположим, что длина просматриваемого нами блока не превышает 255 байтов, так что счетчик BC будет храниться только в регистре C. тогда мы могли бы написать:

```
SEARCH      CPI
            JR Z,FOUND
            INC C
            DEC C
            JR NZ,SEARCH
NOT FOUND   ..... ....
            ....
FOUND       ..........
```

SEARCH — поиск; FOUND — найдено; NOT FOUND — не найдено.

Очевидно, если бы длина блока превышала 255 байтов, нужна была бы другая программа. Обратите внимание на применение команд INC и DEC для проверки условия C = 0. Эти две команды требуют по одному только байту каждая, и поскольку обе они действуют на флаг нуля, результат их действия состоит в том, что флаг устанавливается только если «C» был первоначально равен нулю. Еще одно преимущество состоит в том, что при таком написании программы не изменяются никакие другие регистры.

Теперь мы могли бы также захотеть просмотреть блоки памати начиная с вершины, а не с конца, и поэтому у нас имеется команда:

```
CPD
```

Которая по-русски читается «сравнить и уменьшить». Уменьшение, конечно, относится к HL, а результат для BC — тот же самый!

Еще большими возможностями, чем эти две команды, обладают следующие истинные титаны в мире команд:

```
CPIR
CPDR
```

Они читаются: «сравнить, увеличить и повторить» и «сравнить, уменьшить и повторить».

Эти двухбайтовые команды обладают невероятными возможностями: они позволяют ЦП автоматически продолжить просмотр блока памяти до тех пор, пока либо не будет найдено совпадение, либо не будет достигнут конец блока! (Естественно, нам нужно указать A, HL и BC перед началом, но даже при этих условиях это — невероятно экономичный способ записи)

Поскольку завершение работы этой команды происходит в двух возможных случаях (а именно, при найденном совпадении в середине блока и при отсутствии совпадений вообще), нам необходимо обеспечить наличие некоторого текста в конце, чтобы проводить различие между этими двумя возможностями.

Вам, однако, следует знать, что вне зависимости от скорости выполнения программ на машинном языке CPIR и другие аналогичные команды могут отнимать очень много времени.

Команда CPIR, например, затрачивает 21 цикл на поиск каждого байта. Конечно, каждую секунду выполняется 3 500 000 циклов, но даже с учетом этого поиск среди 3 500 байтов требует 1/50 секунды.

Это может показаться вам не очень длительным временем, но если учесть, что выдача изображения на дисплей осуществляется каждые 1/50 секунды или около того, вы поймете, что это может оказаться важным.

Остальные блочные операции строятся вокруг идеи перемещения информации.

Вот они:

LDI LDIR
LDD LDDR

Очевидно, они относятся к семейству «загрузок» и читаются как:

— Загрузить и дать приращение; — Загрузить, дать приращение и повторить; — Загрузить и уменьшить; — Загрузить, уменьшить и повторить;

Разберем сначала самую простую из них, «LDI» — это на самом деле комбинация следующего набора действий:

Загрузить (HL) в (DE), Дать приращение DE, HL, Уменьшить BC.

Обратите внимание, что это — единственная команда, загружающая из одной ячейки памяти в другую без необходимости загрузки сначала в регистр.

Применение регистра «DE» в качестве адреса-цели очень разумно, так вы никогда не забудете, в каком регистре содержится адрес-цель! (в оригинале используется мнемоника: «DE» DE-STINATION (адрес-цель) (примеч. Пер.))

Симметричная этой команде LDD совершенно совпадает с этой, за исключением того, что в процессе загрузки HL и DE уменьшаются. Разница между LDI и LDD оказывается важна в том случае, когда два блока (тот, где информация находится, и тот, в который она направляется) пересекаются.

Предположим, мы применяем эту команду в прикладной задаче текстовой обработки и хотим удалить слово из предложения:

THE BIG BROWN DOG JUMPED OVER THE FOX.
(большая коричневая собака перепрыгнула через лису).
1 3 5 7 9 1 3 5 7 9 1 3 5 7 9 1 3 5 7 9

Если мы хотим удалить слово «BROWN» (коричневая), то нам нужно только сдвинуть остальную часть предложения влево на 6 литер.

DE = цель = литера 9
HL = источник = литера 15
BC = счетчик = 24 литеры.

Давайте начнем с LDI: после первой команды у нас будет:

Первоначально= THE BIG BROWN DOG JUMPED OVER THE FOX.
Сдвиг на 1 литеру: D<———D
новое предложение= THE BIG DROWN DOG JUMPED OVER THE FOX.
В HL = 10, DE = 16, BC = 23.

После двух следующих команд:

THE BIG DOGWN DOG JUMPED OVER THE FOX.

И после того как все команды выполнены:

THE BIG DOG JUMPED OVER THE FOX.E FOX.

(если бы мы хотели, чтобы часть предложения после точки была заменена пробелами, то этого можно было бы достигнуть за счет добавления пробелов в конце первоначального предложения и увеличения BC, скажем, до30).

Если мы теперь решили обратить этот процесс и возвратить слово «BROWN» в предложение, то нам нужно просто снова применить команду «LDI», поскольку мы забьем информацию, которую мы хотим сдвинуть, например:

HL = источник = литера 9
DE = цель = литера 15
BC = счетчик = 24 литеры

После выполнения одной команды у нас будет:

первоначально = THE BIG DOG JUMPED OVER THE FOX.E FOX.
Сдвиг на литеру D———>D
новое предложение = THE BIG DOG JUDPED OVER THE FOX.E FOX.

После выполнения 6 команд мы получим:

THE BIG DOG JUDOG JUVER THE FOX.E FOX.

Пока все хорошо. Но после выполнения следующих трех команд получим:

THE BIG DOG JUDOG JUD OG THE FOX.E FOX.

Трудность состоит в том, что мы забили информацию, которую мы котели перемещать. Вы можете проверить это, пытаясь переместить по одному символу вручную.

Поэтому лучше использовать команду «LDI» с регистром DE, указывающим на конец предложения. Это гарантирует, что информация не будет забита при перемещении. (в оригинале, очевидно, опечатка, WOLL вместо WILL. (примеч. Пер.))

Команды «LDIR» и «LDDR» обладают еще большими возможностями, они могут быстро перемещать тысячи байтов.

Упражнение

Напишите короткую программу для перемещения 32 байтов из части памяти, занятой ПЗУ, на экран. Обратите внимание, как организованы первые 32 байта экрана. Затем попробуйте переместить 256 байтов, а затем — 2048 байтов.

Более редко используемые команды Z80

Обмен регистрами

Мы коротко рассмотрели в первых нескольких главах идею ЦП, имеющего набор перчаток, которые он может снимать и надевать, и таким образом хранить информацию в таком месте, которое более доступно, чем ячейка памяти.

Вы, должно быть, помните, что вы не можете манипулировать этими альтернативными регистрами, и аналогия с перчатками в этом плане очень ценна. Хотя они и сохраняют свою форму, сами по себе они не могут выполнять арифметические действия или счет.

Первая команда такова:

```
EX AF,AF'
```

Она делает в точности то, что подсказывает ее название: «обменять пары регистров AF и AF» «. Используя аналогию с перчатками, мы сказали бы: »переодеть перчатку с пары рук AF «. Иными словами, надеть запасной набор перчаток AF — вы помните, что запасной набор всегда обозначается AF».

Следующая общая команда обмена перчаток такова:

```
EXX
```

Эта команда переодевает перчатки на всех остальных 8-битовых регистрах следующим образом:

```
B C          B' C'
D E    <—>   D' E'
H L          H' L'
```

Поэтому это — очень мощная команда, но именно ее возможности делают ее ограниченной в применении. Так происходит потому, что она действует на все регистры сразу, и невозможно оставить ни одно значение. (за исключением регистра «А», на который «EXX» не действует).

Единственный способ преодолеть эту трудность — написать короткую программу такого типа:

```
PUSH HL
EXX
POP HL
```

Это означает, что вы запомнили значения BC, DE и HL в наборе альтернативных регистров, но по-прежнему оставляем для работы HL.

Последняя команда из этой группы на самом деле не относится к типу «переодевания перчаток»:

```
EX DE,HL
```

В этой команде задается перекрестный обмен содержимым между регистрами.

HL и DE.

Эта команда на самом деле очень полезна, поскольку, как мы видели, HL — привилегированная пара регистров во многих прикладных задачах, и есть ситуации, когда нужное нам значение для обработки находится в DE.

Установка и сброс битов

До сих пор все команды, с которыми мы работали, включали обработку 8-битовых или 16-битовых чисел.

Группа «установка и сброс битов» позволяет нам обрабатывать отдельные пальцы на руке ЦП (отдельные биты регистров) и/или содержимое ячеек памяти. Из-за очень трудоемкого характера работы с отдельными битами это — не очень широко используемая группа команд.

Более того, как правило, установка отдельного бита в регистре или ячейке памяти занимает даже больше, чем изменение или чтение всех 8 битов этой ячейки памяти или регистра.

Тем не менее, есть ситуации, когда вам необходимо знать, установлен или сброшен бит в середине байта, или даже установить бит. Обратите, тем не менее, внимание, что установка и сброс битов может выполняться с помощью логических операторов.

Группа команд «установки и сброса битов» позволяет «включить» или «выключить» по желанию произвольный бит, или даже просто посмотреть на заданный бит, чтобы проверить, в каком он состоянии.

Давайте посмотрим на первый набор команд:

```
SET N,R
SET N,(HL)
SET N,(IX+Д)
SET N,(IY+Д)
```

Команда «SET» «включает» (т.е. =1) бит с номером «N» (в обозначениях 0—7) в регистре «R» или заданной ячейке памяти.

Никакие флаги не изменяются.

Группа команд «RESET» действует на ту же в точности группу регистров или ячеек памяти, но вместо «включения» битов она их «выключает» (т.е. = 0).

Команды «BIT» на самом деле нужно читать «BIT?», поскольку функция этой команды состоит в проверке содержимого указанного бита.

Никаких изменений в регистры или память не вносятся, но флаг нуля изменяется в соответствии с состоянием проверяемого бита.

Если бит = 0, то флаг нуля устанавливается (= 1).

Если бит = 1, то флаг нуля сбрасывается (= 0).

Это на первый взгляд может показаться запутанным, но нужно посмотреть на это следующим образом: если бит равен нулю, то устанавливается флаг нуля; если бит установлен, то, естественно, флаг нуля подниматься не будет.

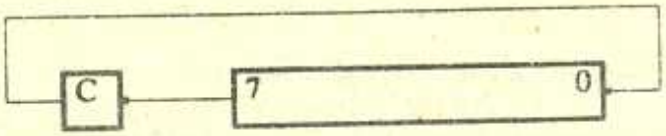
Сдвиги и циклические сдвиги

Вы можете сдвигать регистры влево, вправо, как захотите.

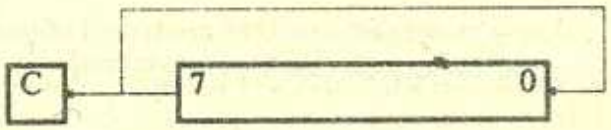
Сложность состоит в том, чтобы различать различные сдвиги и циклические сдвиги с тем, чтобы знать, какие и когда нужно использовать, и помнить, что бит переноса часто можно рассматривать в качестве 9-го бита регистров. (т.е. бит переноса является восьмым битом, если биты пронумерованы от 0 до 7).

Некоторые команды циклического сдвига захватывают бит переноса (в качестве девятого бита), так что циклический сдвиг в целом составляет цикл из 9 битов.

Например, давайте рассмотрим команду «RLA» (смысл каждой команды будет объяснен ниже в этой же главе):



Другие циклические сдвиги захватывают только 8-битовый цикл, хотя флаг переноса изменяется в соответствии со значением бита, которому приходится «проделывать кружной путь». Примером может служить команда «RLC A»:



Это означает, что при циклическом сдвиге влево, как, например, показано выше, содержимое бита 0 перемещается в бит 1, бита 1 — в бит 2 и так далее, а содержимое бита 7 перемещается и в бит 0, и в бит переноса. Сравните это с командой «RLA», описанной выше, где бит 7 перемещается в бит переноса, а бит переноса — в бит 0.

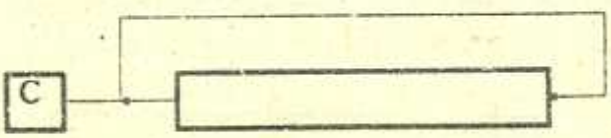
Циклические сдвиги влево

В основном есть два типа циклических сдвигов влево:

Циклический сдвиг влево регистров (циклический сдвиг влево круговой — «круговой» означает, что цикл охватывает только 8 битов, как в команде RLC A, описанной выше) — это 9-битовый сдвиг влево, аналогичный показанному выше для «RL A».

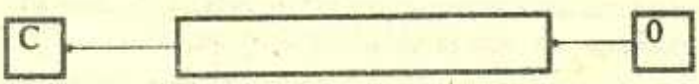
RL A — «циклический сдвиг влево накопителя»;
RL R — «циклический сдвиг влево регистра R».

RLC A — циклический сдвиг влево круговой регистра «A»; RLC R — циклический сдвиг влево круговой регистра «R»; RLC (HL) — циклический сдвиг влево круговой регистра (HL); RLC (IX+Д) — циклический сдвиг влево круговой (IX+Д); RLC (IY+Д) — циклический сдвиг влево круговой (IY+Д).



Помимо этих двух команд циклического сдвига влево, есть еще команда сдвига влево, но она может обрабатывать только регистр «A».

SL A, сдвиг накопителя влево:

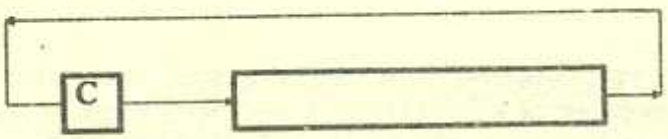


Отличие этой команды состоит в том, что содержимое бита переноса теряется, а нулевой бит заполняется нулем. По существу это сводится к умножению «A» на 2, поскольку ничего в накопитель не переносится. (обдумайте работу команды «SL A» в случае A = 80H).

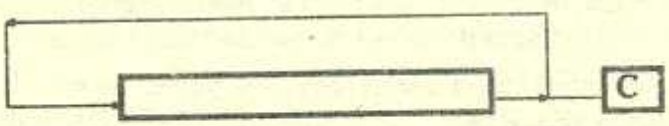
Циклические сдвиги вправо

Вновь у нас два основных режима циклического сдвига, но на этот раз вправо. Вращение вправо охватывает тот же диапазон допустимых ячеек памяти и вращений (Так в оригинале. Возможно, здесь имеется опечатка. Скорее всего нужно: регистров. (примеч. пер.)), что и вращения влево.

RR A — циклический сдвиг накопителя вправо
RR R — циклический сдвиг регистра вправо

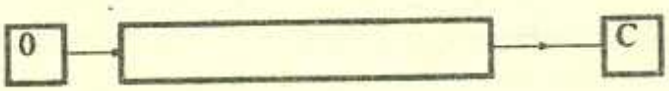


RRC A — циклический сдвиг вправо круговой регистра «A»;
RRC R — циклический сдвиг вправо круговой регистра «R»;
RRC (HL) — циклический сдвиг вправо круговой регистра (HL);
RRC (IX+Д) — циклический сдвиг вправо круговой (IX+Д);
RRC (IY+Д) — циклический сдвиг вправо круговой (IY+Д).



Имеется сдвиг вправо, аналогичный сдвигу влево:

SRL R — логический сдвиг вправо регистра «R»

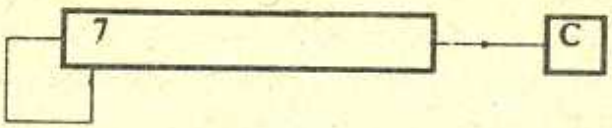


В данном случае это — чистое деление на 2, если мы используем числа без знака (т.е. диапазон чисел, которые мы хотим представлять, составляет 0 — 255).

Поскольку в некоторых прикладных задачах мы применяем соглашение о том, что для задания отрицательных чисел бит 7 устанавливается равным 1 (что дает нам диапазон от -128 до +127), имеется дополнительная (в оригинале, очевидно, опечатка ADDITION SHIFT (примеч. пер.)) команда сдвига вправо, называющаяся

SRA R — арифметический сдвиг вправо регистра «R»

Как видите, это — тоже деление на 2, но оно сохраняет бит знака.



Ввод и вывод

Вопросы ввода и вывода — одни из самых простых в программировании на машинном языке.

Возникают ситуации, когда ЦП необходимо получить информацию из внешнего мира («ЦП — не остров?»), например, с клавиатуры или кассетного магнитофона.

Что касается ЦП, для него это совершенно чуждый мир, и как любой добропорядочный ЦП, он никогда из дома не отлучается. Самое большее, что он может сделать — это открыть дверь и получить доставленные вещи. ЦП не знает и не хочет знать, как работает кассетный магнитофон.

Все, что ему нужно знать, — это к какой двери доставит свои товары посыльный от кассетного магнитофона — у Z80 есть выбор до 256 дверей, но реальное число, доступное конкретному ЦП — это результат решений, принятых производителями аппаратуры. Что касается «синклера», имеются только клавиатура, печатающее устройство и кассетный магнитофон.

Еще одно, о чем ЦП не хочет знать — это как передаются данные. Все, что его касается, — это то, что при вводе и при выводе он получает 8-битовый байт.

Клавиатура и кассетный магнитофон расположены за дверью номер FEH (десятичное 254), так что для получения данных с клавиатуры вы применяете команду:

```
IN A,(FE)
```

Теперь вы можете задать себе вопрос, как 40 клавиш на клавиатуре организованы так, что они представляются 8-битовыми байтами.

Ответ для вас будет скорее всего неожиданным — клавиатура за один раз передает информацию только от 5 клавиш. Значение регистра «А» определяет в момент открытия двери, какая группа из 5 клавиш будет просматриваться!

Клавиатура разделена на 4 ряда, каждый из которых содержит два блока по 5 клавиш:

```
3 =>   1 2 3 4 5      6 7 8 9 0      <= 4
2 =>   Q W E R T      Q U I O P      <= 5
1 =>   A S D F G      H J K L N/L    <= 6
0 => SFT Z X C V      B N M . SPC
```

Легко видеть, что имеется 8 блоков литер, и мы поэтому можем увязать их с 8 битами регистра «А».

Именно так и обстоит дело: все биты регистра «А» устанавливаются в положение «включено» за исключением одного бита, задающего подлежащий чтению блок.

Вы можете представлять это себе как секретный знак-рукопожатие — когда ЦП идет к двери за информацией, рукопожатие определяет, какую часть информации он получит.

Так, чтобы прочитать клавиши из блока «1 2 3 4 5», должен быть выключен бит 3 регистра «А»:

```
A = 1 1 1 1 0 1 1 1 = F7
```

Содержимое посланной с клавиатуры информации поступает в регистр «А» в его младшие биты:

Клавиша «1» —> бит 0 регистра «А»
Клавиша «2» —> бит 1 регистра «А»

Вы можете считать, что информация поступает в регистр «А» сначала с внешних сторон клавиатуры, так что и «0» и «1» попадут в бит «0» регистра «А».

Для некоторых игр вам может понадобиться позволить считывать весь верхний ряд, причем весь его можно читать одной командой (а не двумя, как потребовалось бы при чтении одного блока за раз).

Для этого посыльного обманывают, чтобы он выдал вам сразу две порции информации, например:

A = 1 1 1 0 0 1 1 1 = E7

Обратите внимание, что оба бита «3» и «4» «выключены».

Такое рукопожатие говорит посыльному, что ЦП нужна информация из блоков 3 и 4, и именно ее он и получит. Конечно, две порции информации оказываются перемешанными, так что вы не сможете сказать, была нажата клавиша «0» или «1», например, обе установят бит 0 регистра «А».

Т.е.

«1» или «0» —> бит 0 регистра А
«2» или «9» —> бит 1 регистра А

Это полезно для динамических игр, поскольку позволяет использовать клавиши «5» и «8» в качестве указателя направления движения направо и налево, хотя они и находятся в разных блоках клавиатуры.

Обратите внимание, что если вы применяете команду

IN R,(C),

где регистр С задает используемую вами дверь, то содержимое регистра В определяет, какой блок клавиатуры выбирается.

Другие двери, представляющие для вас интерес, очевидно, двери ввода-вывода с кассетного магнитофона.

Это — дверь FE. Как сказано выше, основная возникающая проблема связана с синхронизацией ввода и вывода данных; проблемы такого типа требуют большого опыта программирования на машинном языке и вычисления времени выполнения каждой команды.

Команда OUT также применяется для генерирования звукового сигнала на «SPECTRUM» и установления цвета окаймления.

На страниц 160 руководства «SPECTRUM» рассматривается команда OUT в языке BASIC, а программирование команды на машинном языке в точности такое же. Иными словами, биты 0, 1 и 2 определяют цвет окаймления, бит 3 посылает импульс на гнезда MIC и EAR, а бит 4 посылает импульс на внутренний громкоговоритель.

Чтобы изменить цвет окаймления, нужно загрузить в регистр А соответствующее значение цвета, а затем выполнить команду OUT (FE),A. Обратите внимание, что это — лишь TEMPORARY (временное) изменение цвета окаймления. Чтобы постоянно изменить цвет окаймления, вы должны выполнить приведенную выше команду OUT и также изменить значение ячейки памяти 23624, представляющую собой переменную BORDER операционной системы (см. страницу 174 руководства «SPECTRUM»).

Причина этого состоит в том, что аппаратура «SPECTRUM» (чип ULA «SPECTRUM») управляет цветом окаймления и она получает информацию о нем, просматривая содержимое этой ячейки памяти. Вы можете предотвратить всю эту волынку ЦП с цветом окаймления только заблокировав все прерывания (команда DI). Обратите внимание, что некоторые подпрограммы в ПЗУ разблокируют прерывания (команда EI).

Самостоятельная генерация звукового сигнала

Вы можете сгенерировать свой собственный звуковой сигнал на «SPECTRUM», но для пользователей, имеющих только 16K памяти с произвольным доступом имеются определенные ограничения, порождаемые особенностями аппаратуры.

Поскольку экран постоянно обновляется, аппаратура регулярно прерывает выполнение заданий Z80 для того, чтобы показать, что находится в файле дисплея. Это выполняется путем снижения напряжения в схеме WAIT.

В результате оказывается невозможным создание какой бы то ни было программы, требующей точной или регулярной синхронизации, поскольку невозможно предсказать результаты синхронизации с помощью этих прерываний WAIT. Конструкция «SPECTRUM» такова, что прерывание Z80 осуществляется при попытке обработки информации, содержащейся в первых 16K памяти с произвольным доступом. Такого прерывания не происходит, если программа и данные, к которым осуществляется доступ Z80, находятся в ПЗУ или верхних 32K памяти.

Обобщить все это в терминах, понятных непосвященным, можно следующим образом: вы можете воспроизводить звуки и шумы с помощью команды OUT на ЭВМ с 16K памяти, но это не будут чистые ноты. (обойти это можно с помощью программы BEEP из ПЗУ. См. главу о средствах «SPECTRUM»).

Для генерации звука вы должны послать импульс для включения громкоговорителя (или (и) на гнездо MIC, если его нужно усиливать). Затем, спустя некоторое время, вам нужно послать другой импульс для его выключения. Затем еще спустя некоторое время вновь включить и т.д...

Таким образом создается звук. Интервал времени между двумя последовательными включениями громкоговорителя определяет чистоту звука. Продолжительность интервала времени, в течении которого громкоговоритель остается включенным, в противоположность интервалу времени между импульсами, может дать вам минимальную степень контроля громкости звука.

Обратите внимание, что вы должны применять для включения и выключения такое значение A, чтобы цвет окаймления не изменялся. В противном случае частотная характеристика полученного звука будет напоминать звуки загрузки.

Упражнение

Напишите программу, имитирующую звук сирены скорой помощи (частота возрастает, а затем снижается). Обратите внимание, что вы должны обеспечить звучание каждой частоты в течение некоторого короткого интервала времени, прежде чем перейти к следующей частоте.

Представление двоично-кодированных десятичных чисел

Двоично-кодированное десятичное представление по английски сокращается как BCD (BINARY-CODED DECIMAL). Это — один из способов представления информации в десятичной форме.

Чтобы закодировать каждую из цифр от 0 до 9 необходимо всего 4 бита, и 6 допустимых кодов не будут в этом представлении использоваться.

Поскольку для кодирования десятичной цифры необходимо 4 бита, в каждом байте можно закодировать две цифры. Это называется двоично-кодированным десятичным представлением. Например, 0000 0000 — двоично-кодированное десятичное представление десятичного числа 00. 1001 1001 — двоично-кодированное десятичное представление десятичного числа 99. Каково будет двоично-кодированное представление чисел «58», «10?

Будет ли «1010 0000» допустимым двоично-кодированным десятичным представлением числа?

Арифметические действия над двоично-кодированными десятичными числами

Эта странная система обозначений для представления чисел может повести к потенциальным проблемам при сложении и вычитании.

Попробуем сложить следующие числа

```
BCD   08     0000 1000
BCD   03     0000 0011
-------------------------
BCD   11     0000 1011
```

Вы заметите, что результат второй операции неверен и является недопустимым в представлении двоично-кодированном десятичным числом. Для компенсации этих трудностей необходимо применять особую команду, «DAA», называемую «десятичная настройка арифметических действий», чтобы исправить результат сложения. (т.е. добавлять 6, если результат превышает 9).

Следующую трудность иллюстрирует тот же пример, будет генерироваться бит переноса из младшего разряда двоично-кодированного десятичного числа (самого первого) в самый левый. Этот внутренний перенос должен быть учтен и добавлен во второй разряд двоично-кодированного десятичного.

Для выявления этого переноса используется «флаг половинного переноса», 'H'

```
LD A,12H       : LOAD LITERAL BCD ''12''
ADD A,24H      : ADD LITERAL BCD ''24''
DAA            : DECIMAL ADJUST RESULT
LD (ADDR),A    : STORE RESULT
```

LOAD LITERAL — загрузить литерал; ADD LITERAL — добавить литерал; DECIMAL ADJUST RESULT — десятичная настройка результата; STORE RESULT — запоминание результата.

При программировании вам вряд ли потребуется двоично-кодированное десятичное представление. Но неплохо знать, что чип Z80 все-таки поддерживает это представление, и команда DAA облегчает жизнь небольшой группе пользователей двоично-кодированного десятичного представления.

Прерывания

Прерывание — это посылаемый микропроцессору сигнал, который может появиться в любое время и, вообще говоря, приостановит выполнение текущей программы (так, что программа даже не узнает об этом).

Z80 предоставляет три механизма осуществления прерываний: запрос шины (BUSRG), немаскируемое прерывание (NMI) и обычное прерывание (INT).

С точки зрения программирования мы рассмотрим только обычное маскируемое прерывание (INT).

Команда DI (блокировка прерывания) применяется для сброса (маскирования), а команда EI (разблокирование прерывания) для установки (размаскирования).

В общем случае обычное прерывание приведет к тому, что текущий счетчик команд будет помещен в стек, а управление (в оригинале возможно опечатка. Перевод дается по смыслу. (примеч. пер.)) с помощью команды RST будет передано на нулевую страницу ПЗУ. Для возвращения из прерывания требуется команда RETI (возврат из прерывания).

При нормальном режиме работы у «SPECTRUM» прерывания разблокированы (EI), и на самом деле программа прерывается 50 раз в секунду. Это прерывание позволяет программе ПЗУ осуществлять сканирование клавиатуры.

Вам может понадобиться заблокировать прерывания в вашей программе, поскольку это ускоряет выполнение. Вы все-таки сможете получать информацию с клавиатуры, при условии использования своей собственной программы для этого. Обязательно разблокируйте прерывания, когда закончите выполнение программы, поскольку иначе система не сможет получать информацию с клавиатуры!

Команда рестарта (RST)

Это — скорее всего «рудименты» чипа 8080, реализованные в целях совместимости. Поэтому вы вряд ли станете применять команду RST в своей программе.

Команда RST выполняет те же самые действия, что и вызов, но позволяет совершать переход только по восьми адресам, расположенным в первых 256 ячейках памяти: 00H, 08H, 10H, 18H, 20H, 28H, 30H, и 38H.

Преимущество команды RST состоит в том, что часто используемые подпрограммы можно вызвать, затрачивая на это всего один байт. Кроме того, команда RST занимает меньше времени, чем команда CALL.

Недостатком команды RST является то, что ее можно применять только для обращения к одной из восьми приведенных выше допустимых ячеек.

Поскольку все эти ячейки расположены в ПЗУ, вы можете воспользоваться этим преимуществом в своей собственной программе. Есть, однако, возможность использовать подпрограммы ПЗУ, если вы знаете, что они делают, и тем самым использовать команды RST.

Вы сможете больше узнать о командах RST из нашей книги «UNDERSTANDING YOUR SPECTRUM», написанной доктором Яном Логаном.

Написание программ для «SPECTRUM»

Планирование вашей программы на машинном языке

Программирование на машинном языке обладает чрезвычайной гибкостью в том смысле, что позволяет вам делать все, что угодно.

Поскольку от всех языков более высокого уровня в конечном счете нужно переходить к машинному языку, все, что вы можете запрограммировать на языке «фортран» или «кобол», или любом другом, можно запрограммировать на машинном языке.

Дополнительным преимуществом будет то, что программа на машинном языке выполняется быстрее.

Эта абсолютная гибкость может, однако, оказаться ловушкой для неосторожного программиста. При наличии такой полной свободы можно делать все, что угодно. В отличие от операционной системы «SPECTRUM» для языка BASIC, например, отсутствуют проверки предложений на допустимость.

Поскольку любые вводимые вами числа будут командами того или иного типа, чип Z80 будет обрабатывать все на свете.

Но даже не беря в расчет проблемы проверки допустимости синтаксиса, нужно отметить, что программирование на машинном языке не накладывает ограничений на используемую вами логику: вы можете выполнять функции, переходы и т.п., которые будут абсолютно недопустимы в любом языке более высокого уровня.

Поэтому огромное значение приобретает самодисциплина при разработке программ на машинном языке. Невозможно преувеличить значение концепции «нисходящего» подхода в программировании в целом, но в особенности это касается программирования на машинном языке.

«Нисходящий» подход заставляет вас разбивать задачу на более мелкие единицы и позволяет проверить логику вашей разработки, на протяжении долгого времени не записывая ни одной строки текста программы.

Предположим, вам захотелось написать программу посадки на Луну:

Самый первый путь мог бы напомнить нечто подобное:

```
INSTR   выдача на экран инструкций
        Переход назад на INSTR, пока не будет нажата клавиша ENTER
DRAW    нарисовать ландшафт, начать движение спускаемого аппарата
        с вершины экрана.
LAND    движение спускаемого аппарата
        Если горючее кончилось, перейти на CRASH
        Перейти назад на LAND, если поверхность не достигнута
GROUND  напечатать поздравления
        Перейти назад на INSTR для следующего прогона
CRASH   напечатать соболезнования по поводу неудачной посадки
        Перейти назад на INSTR для следующего прогона.
```

Обратите внимание, что вся эта «программа» написана на русском языке. На этом этапе не принималось никаких решений, будет ли программа писаться на языке BASIC или на машинном языке. Такое решение и не нужно принимать — концепция программы посадки на Луну не зависит от способа записи ее текста.

Теперь наступает этап логической проверки. Вы выполняете роль ЭВМ и смотрите, все ли возможности, которые вы хотели включить в программу, имеются в наличии.

Нет ли переходов на объекты, которые вы хотели написать, но забыли? Все ли есть? Нет ли избыточных программ? Не следует ли некоторые объекты перенести в подпрограммы?

Давайте снова посмотрим на «программу» — ох-ох-ох! — мы забыли как-нибудь закончить программу!

Описанная выше логика может быть прекрасной для некоторых прикладных задач, таких как игровая ЭВМ, но в своей программе вам может прийти в голову, что неплохо бы иметь возможность выключить работу программы.

Теперь мы изменим последнюю часть программы следующим образом:

```
GROUND  напечатать поздравления
        Перейти на CRASH
        напечатать соболезнования по поводу неудачной посадки
FINISH  спросить игрока, нужно ли закончить
        Если нет, перейти на INSTR
        Если да, STOP
```

Обратите внимание, что мы использовали метки для описания определенных строк программы. Метки — очень ценный аппарат, особенно если вы будете выбирать их краткими и содержательными.

Когда этот уровень закончен, вы перемещаетесь на уровень глубже, чтобы проделать то же самое с одной из строк или модулей, приведенных выше.

Именно поэтому этот подход называется нисходящим.

Например, мы можем следующим образом расписать модуль «FINISH», приведенный выше:

```
FINISH      Очистить экран
            Напечатать: «хотите ли закончить?»
            Опросить клавиатуру в ожидании ответа
            Если ответ = да, то закончить
            Перейти на INSTR
```

Еще одно преимущество нисходящего подхода состоит в том, что вы можете тестировать и выполнять конкретный модуль автономно, так что он будет отлажен для включения в окончательный текст программы.

Давайте спустимся еще на один уровень и посмотрим на строку «Очистить экран» более подробно.

На этом этапе нам нужно решить, на каком языке мы будем писать программу, и давайте выберем машинный язык «синклера».

Если бы вы писали программу на языке BASIC, то вам было бы достаточно написать:

```
900 CLS
```

Но на машинном языке это простое предложение «очистить экран» может оказаться обманчивым.

Поэтому мы могли бы сделать что-то в таком роде:

```
CLEAR       найти начало экрана
            Заполнить следующие 6144 позиции пробелами
```

Мы все еще не написали ни строчки текста программы, но, очевидно, подход основан на машинном языке. Давайте посмотрим более пристально, что должна делать эта программа очистки экрана и что она делает на самом деле.

Вы, возможно, помните из руководства по «SPECTRUM», что экран состоит из 6144 ячеек и что есть еще 768 ячеек, описывающих атрибуты экрана: цвет бумаги, цвет чернил и т.п.

Приведённое выше краткое описание программы, конечно, очистит часть экрана, но никак не повлияет на файл атрибутов. Если не у всех позиций экрана совпадает цвет бумаги или если позиции некоторых литер имеют включенные атрибуты мигания или яркости, то ясная программа очистки экрана, приведенная выше, окажется явно неадекватной.

Нам придется также обработать и файл атрибутов. (обратите внимание, насколько сложнее оказываются некоторые задачи на машинном языке, чем на языке BASIC)

Поэтому нам нужно расширить программу до следующего вида

— Найти начало экрана
— Заполнить следующие 6144 байта пробелами
— Найти начало файла атрибутов
— Заполнить следующие 768 байтов требуемыми значениями атрибутов бумаги (чернил)

Следующий нижележащий уровень — это уже тот, на котором вы должны, наконец, писать текст программы, так что давайте посмотрим, как экран заполняется пробелами:

```
CLEAR LD HL,SCREEN   : SCREEN START
LD BC,6144           : BYTES TO CLEAR
LD D,0               : D=BLANK
LOOP LD (HL),D       : FILL BLANK
INC HL               : NEXT POSITION
DEC BC               : REDUCE COUNT
LD A,B
OR C                 : TEST IF BC=0
JR NZ,LOOP           : AGAIN IF NOT END
```

SCREEN START — начало экрана; BYTES TO CLEAR — очищаемые байты; BLANK — пробел; FILL BLANK — заполнение пробелом; NEXT POSITION — следующая позиция; REDUCE COUNT — уменьшение счетчика; TEST — проверка; AGAIN IF NOT END — повторение, если не достигнут конец.

Теперь вы достаточно легко можете работать с программами такой длины и таким способом строить достаточно большие программы.

Кстати, вы теперь без сомнения понимаете, почему программы на машинном языке часто так велики по объему и почему люди изобрели программы на языках высокого уровня!

Упражнения

Чтобы написать любую конкретную программу, есть всегда больше одного способа, так что давайте рассмотрим простую программу очистки экрана, написанную выше.

Для этого есть несколько разных способов.

Упражнение 1

Можете ли вы придумать способ, позволяющий в цикле опробелить 6144 позиций без применения регистра BC, а только с помощью регистра B так, чтобы мы могли пользоваться командой «DJNZ»?

Упражнение 2

Можете ли вы придумать способ, позволяющий опробелить 6144 позиций с помощью более мощной команды «LDIR»?

Тщательно продумайте, что делает команда «LDIR»: не всегда необходимо иметь где-то еще 6144 пробельных позиций!

Ответы

Есть несколько «правильных» ответов, единственная проверка — будет ли это работать? Иными словами, делает ли программа то, что вам нужно?

С помощью «DJNZ»:

```
CLEAR     LD HL,SCREEN
          LD A,0
          LD B,24          : SET B=24
BIGLOOP   PUSH BC          : SAVE VALUE
          LD B,A           : SET B=256
LITLOOP   LD (HL),A        :
          INC HL           : FILL IN 256 BLANKS
          DJNZ LITLOOP
          POP BC           : GET BACK VALUE OF B
          DJNZ BIGLOOP     : DO IT UNTIL END
```

SET — установить; SAVE VALUE — запомнить значение; FILL IN 256 BLANKS — заполнить 256 пробелов; GET BACK VALUE OF B — получит назад значение B; DO IT UNTIL END — выполнять до достижения конца.

Мы смогли использовать 24 раза по 256 (=6144) для очистки экрана. Нужно отметить следующее: Мы можем задать B = 0, чтобы пройти цикл DJNZ 256 раз. (почему?)

Эта процедура обычно не будет применяться в программе, если мы только не станем использовать регистр C для других целей.

Применение LDIR:

```
CLEAR     LD HL,SCREEN     : SOURCE
          PUSH HL
```

```
POP DE
INC DE          : DEST = HL + 1
LD BC,6144      : HOW MANY
LD (HL),0       : LAT POS = 0
LDIR            : MOVE IT
```

SOURCE — источник; HOW — сколько; LAT — первая; MOVE — перемещение.

Обратите внимание, что мы получили DE = HL + 1, задавая DE = HL и давая приращение DE. Это можно сделать проще, загружая значение SCREEN + 1 в DE непосредственно, но для этого требуется на один байт больше!

Причина, по которой эта команда LDIR срабатывает, состоит в том, что используется факт, что в процессе обработки данные перезаписываются в блок. Здесь происходит применение с положительным результатом задачи, рассмотренной нами в главе о перемещении блоков.

Если просуммировать потребную память, то при первом методе требуется 14 байтов, при втором — 16 байтов, а при последнем — 13 байтов.

Средства «ZX SPECTRUM»

• Настало время рассмотреть средства вашего «ZX SPECTRUM», полезные при разработке для него программ на машинном языке.

Ввод — клавиатура

Что касается ввода информации в «SPECTRUM», мы будем игнорировать ввод с кассетного магнитофона и сконцентрируемся на клавиатуре.

Клавиатура — единственный источник информации, предоставляющий связь в реальном масштабе времени. Она может динамически влиять на выполнение любой программы, как операционной системы в ПЗУ, так и пользовательской программы в памяти с произвольным доступом.

Логически мы можем рассматривать клавиатуру как двумерную матрицу с восемью рядами и пятью столбцами, как в приложении А.

Каждое из 40 пересечений представляет клавишу клавиатуры. В нормальном состоянии (когда они не нажаты) они всегда в хорошем настроении, т.е. пересечение устанавливается равным 1.

При нажатии конкретной клавиши «нажатое» пересечение, соответствующее этой клавише, будет сброшено в плохое настроение, т.е. 0.

Зная связь между клавиатурой и этой внутренней матрицей представления, мы можем вывести логический способ проверки нажатия клавиши, который можно применять в программировании на машинном языке.

В языке BASIC при сканировании клавиатуры нам нужно задать адрес той конкретной половины ряда клавиатуры, где располагается нужная клавиша, прежде чем использовать функцию IN, как описано в главе 23 (с. 160) руководства «SPECTRUM»

Аналогичным образом в программе на машинном языке мы должны загрузить в накопитель значение, соответствующее адресу полуряда клавиш, который мы хотим проверить. Требуемое значение для каждой половины ряда приводится в самой левой колонке таблицы в приложении А.

Например, для «H — ENTER» (полуряда) мы загружаем в регистр А значение LD A,BFH. Затем значение А будет использоваться для поиска байта, содержащего состояние той конкретной половины ряда клавиш и возврата в А при задании команды INPUT.

Например, используется порт FEH

```
IN A,(FEH).
```

Поскольку в половине ряда имеется пять клавиш, нас интересуют только пять младших битов байта, возвращаемого в регистре A.

Если в этой половине ряда никакие клавиши нажаты не были, то значение пяти младших битов будет (2**4 + 2**3 + 2**2 + 2**1 + 2**0 т.е. 16 + 8 + 4 + 2 + 1 = 31).

Регистр A = XXX11111, когда нет нажатых клавиш.

Если мы хотим проверить, нажат ли самый первый бит, то мы проверяем, сброшен ли он.

Есть два способа проверить это:

1. С помощью команды проверки бита, например BIT 0,A Если бит сброшен (не установлен), то будет установлен флаг нуля.

2. С помощью команд логического и AND 1 Если бит сброшен (не установлен), то результат будет нулевым и флаг нуля будет установлен.

Первый способ проще, поскольку конкретный бит, подлежащий проверке, указан непосредственно в команде проверки бита. Его недостаток состоит в том, что если нам нужно проверить две клавиши в этой половине ряда, нам придется применять две команды проверки бита, и, возможно, два относительных перехода.

Например, чтобы проверить биты 0 и 1 с помощью первого способа:

```
BIT 0,A              : TEST BIT 0 OF A SET OR NOT
JR Z,NPRESS          : JUMP IF NOT PRESSED
BIT 1,A              : TEST BIT 1 OF a SET OR NOT
JR Z,NPRESS          : JUMP IF NOT PRESSED
.
.
DO WHATEVER IF BOTH ARE PRESSED
.
.
NPRESS .
. .
```

TEST BIT 0 OF SET OR NOT — проверить, установлен или нет бит 0 регистра A; JUMP IF NOT PRESSED — переход, если не нажата; TEST BIT 1 OF SET OR NOT — проверить, установлен или нет бит 1 регистра A; DO WHATEVER IF BOTH ARE PRESSED — выполнить то, что следует, если обе клавиши нажаты.

Второй способ проверки с помощью логического и требует несколько больше логических ухищрений. Для проверки бита 0 мы используем «AND 1»; для проверки бита 1 — «AND 2»; для проверки бита 2 — «AND 4» и т.д.

Для проверки двух клавиш мы применяем «AND x», где x — сумма значений, используемых для проверки каждой из клавиш в отдельности.

Например, чтобы проверить, что бит 0 и бит 1 регистра A установлены:

```
AND 1                : TEST BOTH BIT 0 AND BIT 1 IS SET
CP 1                 : TEST IF BOTH SET
JK NZ,NBOTH          : JUMP IF NOT BOTH PRESSED
```

TEST BOTH BIT 0 AND BIT 1 IS SET — проверить, что оба бита 0 и 1 установлены; TEST IF BOTH SET — установлены ли; JUMP IF NOT BOTH PRESSED — переход, если не обе клавиши нажаты.

Чтобы проверить, что установлен один из битов 0 или 1 регистра A.

```
AND 3                : TEST ELTHER BIT 0 AND BIT 1 IS SET
JR Z,NOTONE          : JUMP IF NOT ONE IS PRESSED
```

TEST ELTHER BIT 0 AND BIT 1 IS SET — проверить, что один из битов 0 или 1 установлен; JUMP IF NOT ONE IS PRESSED — переход, если ни одна клавиша не нажата.

Упражнение

Чтобы подвести итог всему, что мы узнали о клавиатуре, попробуйте запрограммировать на машинном языке программу для вашего «SPECTRUM», выполняющую прерывание по нажатию клавиши (ENTER).

Вам понадобится:

А) проверить адрес ряда, который нужно загрузить в регистр А;

Б) послать его на входной порт FEH;

В) проверить бит, устанавливаемый клавишей (ENTER).

Вывод — дисплей с телевизионным экраном

Дисплей с телевизионным экраном — основное средство вывода для ЭВМ при общении с пользователем.

Приводимая ниже программа на машинном языке показывает способ организации памяти экрана в «SPECTRUM»:

```
21 0040     LD HL,4000H     : LOAD HL WITH START OF DISPLAY FILE
36FF        LD (HL),FFH     : FILL THET SCREEN LOCATION
110140      LD DE,4001H     : LOAD DE WITH BYTE IN DISPLAY
010100      LD BC,1         : BC CONTAINS NUMBER OF BYTES TO BE
                            : TRANSFERRED
EDB0        LDIR            : MOVE A BLOCK LENGTH BC FROM (HL)
                            : TO (DE)
C9          RET             : END OF PROGRAM
```

LOAD HL WITH START OF DISPLAY FILE — загрузить в HL начало файла дисплея; FILL THAT SCREEN LOCATION — заполнить эту ячейку экрана; LOAD DE WITH BYTE IN DISPLAY — загрузить в DE следующий байт дисплея; BC CONTAINS NUMBER OF BYTES TO BE TRANSFERRED — в BC содержится число передаваемых байтов; MOVE A BLOCK LENGTH BC FROM (HL) TO (DE) — переместить блок длиной BC из (HL) в (DE); END OF PROGRAM — конец программы.

Загрузите приведенную выше программу в свой «SPECTRUM» и выполните программу на машинном языке. В том виде, как она написана выше, из (HL) в (DE) будет перемещен всего один байт.

Теперь давайте изменим четвертую строчку так: LD BC,31 (011F00). Вас, возможно, удивит, какие будут высвечены первые 32 байта на экране. Обратите внимание, вверху экрана будет проведена очень тонкая полоса. Первые 32 байта экранной памяти относятся к первому байту каждого из первых 32 символов.

Теперь изменим эту строку так: LD BC,255 (01FF00). Вновь вы, возможно, удивитесь. Следующий байт после 32-го не окажется во втором ряду точек на экране! Это — первый байт 32-го символа! И так далее вплоть до 256-го символа.

Сможете ли вы предсказать, куда попадет следующий байт? Измените эту строку так: LD BC,2047 (01FF07) и выполните программу. Вы обнаружите, что заполненной оказалась только верхняя треть экрана.

Вы можете поэкспериментировать с этим, пользуясь разными значениями BC вплоть до LD BC,6143 (01FF17). Таким способом вы можете посмотреть, как «SPECTRUM» организует экран.

Экранная память на самом деле разделена на три части.

1. Память с 4000H по 47FFH <——> первые восемь строк.

2. Память с 4800H по 4FFFH <——> вторые восемь строк.

3. Память с 5000H по 57FFH <——> третьи восемь строк.

Но не только это, вы еще вспомните, что в «SPECTRUM» каждая литера состоит из восьми 8-битовых байта, что составляет 64 точки.

Например, для литеры «!» представление имеет вид:

```
  0 00000000  0H
 16 00010000 10H
 16 00010000 10H
 16 00010000 10H
 16 00010000 10H
  0 00000000  0H
 16 00010000 10H
  0 00000000  0H
```

Структура памяти экрана дисплея «SPECTRUM» такова, что первые 256 байтов с 4000H по 40FFH соответствуют первым байтам каждой из 256 8-байтовых литер первых восьми строк.

Далее следующие 256 байтов ячеек памяти с 4100H по 42FFH соответствуют вторым байтам каждой 256 8-байтовых литер первых восьми строк и т.д.

Таким образом, расположение в памяти восьми байтов, соответствующих первой литере на экране, будет следующим:

```
1ST   BYTE 4000H
2ND   BYTE 4100H
3RD   BYTE 4200H
4TH   BYTE 4300H
5TH   BYTE 4400H
6TH   BYTE 4500H
7TH   BYTE 4600H
8TH   BYTE 4700H
```

1ST BYTE — 1-й байт

Странно, не так ли? Но приходится принимать «SPECTRUM» таким, как его сконструировали.

Сможете ли вы записать восемь байтов, соответствующих 31-й литере третьей строки экрана? Вы можете обратиться к приложению В, карте памяти экрана. (405EH, 415EH, 425EH..... 475EH) в соответствии с принятым нами способом выдачи на экран ячеек памяти, соответствующие первой литере второй группы из восьми строк, будут такими:

4800H, 4900H, 4A00H, 4B00H, 4C00H, 4D00H, 4E00H, 4F00H

Аналогичным образом, первая литера третьей группы из восьми строк отображается восьмью байтами в следующих ячейках:

5000H, 5100H, 5200H, 5300H, 5400H, 5500H, 5600H, 5700H.

В применении машинного языка, тем не менее, есть определенные преимущества. Очевидные трудности стоит преодолевать. Вот тривиальный пример из языка BASIC: если вы попытаетесь выполнить команду PRINT для вводной части экрана (нижние две строки), то система BASIC этому резко воспротивится. Но на машинном языке у вас имеется полный доступ ко всему экрану.

Если вы более внимательно присмотритесь к организации экрана дисплея, вы увидите, что старший байт первого байта (10BFB) каждой литеры определяет, к какой из трех групп памяти литера относится. Например:

если 40H = (H0BFB (41H	литера находится в первой группе из восьми строк
Если 48H = (H0BFB (49H	литера находится во второй группе из восьми строк
Если 50H = (H0BFB (51H	литера находится в третьей группе из восьми строк

Помимо этого три младших бита HOB (HIGH ORDER BYTE. старшего байта) определяют, к какому из восьми байтов литеры он принадлежит.

Ситуация несколько проясняется? Посмотрите приложение В и постарайтесь понять связь между ячейками памяти и экраном дисплея (если она имеется!).

Рассмотрим следующий пример:

Предположим, нам задан адрес 4A36H. Старший байт адреса будет 4AH, так что:

1) Мы знаем, что он находится в пределах памяти экрана дисплея, поскольку его значение находится между 40H и 58H;

2) Его двоичное представление имеет вид 01001010;

3) По младшим трем битам мы знаем, что он принадлежит третьему байту позиции литеры на экране;

4) Если мы сделаем младшие три бита равным нулю, то значение HOB будет равно 48H. Таким образом мы знаем, что он принадлежит второй группе из восьми строк, т.е. средней порции экрана.

Мы можем прийти к выводу, что заданный байт относится к третьему байту литеры в средней порции памяти дисплея.

Какой литере из серединной порции этот байт принадлежит? Для ответа на этот вопрос нам потребуется знать значение младшего байта адреса.

Мы знаем, что младший байт адреса равен 36H. Так что адрес относится к литере 36H (48 + 6), т.е к 54 позиции, считая от первой литеры серединной позиции:

```
                    : в зависимости от того, чему равнялся H:
                    : 48H, 50H или 50H (так в оригинале, очевидно,
                    : опечатка (примеч.пер.))
ADD A,58H           : преобразование в память атрибутов
LD H,A              : в HL — адрес атрибута,т.е H = 58H,59H или
                    : 60H. L остается неизменным!!!
```

Вам, возможно, придется немного обдумать все это!

Ответ на первое упражнение связан со способом работы программы:

```
1ST CHAR OF 1ST SCREEN SECTION=4000H ATTRIBUTE
ADDRESS=5800H
1ST CHAR OF 2ND SCREEN SECTION=4800H ATTRIBUTE
ADDRESS=5900H
1ST CHAR OF 3RD SCREEN SECTION=5000H ATTRIBUTE
ADDRESS=5A00H

2ND CHAR. OF 1ST SCREEN SECTION=4801H ATTRIBUTE
ADDRESS=5801H
    ETC. ...
        ETC. ....

1ST CHAR. OF 1ST SCREEN SECTION — 1-я литера... -й части экран
ATTRIBUTE
ADDRESS — адрес атрибута
```

Это должно все несколько прояснить!

Звуковой сигнал

Еще одно средство связи в реальном масштабе времени, предоставляемое вашей микро-ЭВМ «SPECTRUM» — звуковой сигнал. Было бы глупо не воспользоваться этим средством в полной мере.

В машинном языке «спектрум» есть два основных способа генерирования звука:

1. Посылка сигналов на выходной порт 254 для кассетного магнитофона в течение определенного промежутка времени с помощью команды OUT 254. Например:

```
OUT (254),A
```

2. Установить определенные значения в HL, DE и вызвать программу звукового сигнала из ПЗУ для генерации сигнала.

Входные параметры:

DE — продолжительность в секундах * частота
HL — (437 500 / частота) — 30,125

Потом

```
CALL 03B5H
```

Преимуществом первого способа генерации звука является отсутствие обращений к ПЗУ. Он выполняется более быстро, но.... Всегда есть но!

Поскольку АЛУ (в оригинале ULA — возможно, опечатка. Переводится по смыслу (примеч. пер.)) постоянно обращается к первым 16К памяти с произвольным доступом для выполнения вывода на экран, ваша программа, если она размещена в первых 16К, будет часто подвергаться кратковременным прерываниям.

Если программа генерирует звуковой сигнал, то звук будет издаваться в виде непредсказуемых по продолжительности гудков. Один из способов преодоления этой трудности состоит в перемещении той части программы, которая генерирует звук, в область больших значений адресов памяти, если у вас ЭВМ с объемом памяти 48К.

Если же у вас нет ЭВМ с объемом памяти 48К, то вы все-таки можете генерировать звук этим методом, но это не будет «чистый звук». Вам придется применять второй способ генерации звука (с помощью вызова программы из ПЗУ), чтобы добиться нужного результата.

Обратите внимание, что при посылке значений на порт вывода 254 они будут также влиять на цвет окаймления, и включать MIC, а также громкоговоритель, в зависимости от посылаемого значения.

С другой стороны, программа из ПЗУ для генерирования звука по существу позволяет вам применять из своей программы на машинном языке команду BEEP. Вы можете считать, что в паре регистров содержится значение продолжительности звукового сигнала, а в HL — значение частоты. Поэкспериментируйте с различными значениями HL и DE, пока не получите нужный вам звук.

Ограниченность этого метода, конечно, состоит в том, что вы не можете выйти за пределы того диапазона звуков, которые дает возможность издавать команда BEEP.

Введение в мониторные программы на машинном языке

Мониторная программа EZ CODE

Это — мониторная программа на машинном языке, позволяющая вам:

1. Вводить свой программный модуль, написанный на машинном языке либо в полностью ассемблированном виде, либо в полуассемблированном виде, когда все относительные переходы и абсолютные переходы выражаются через номера строк.

2. Распечатывать исходный вводимый программный модуль.

3. Делать дамп входного программного модуля по заданному адресу памяти.

4. Просматривать ячейки памяти из определенного диапазона.

5. Запоминать либо «исходный модуль», либо дамп программы, полностью переведенной на машинный язык.

6. Загружать записанную «исходную программу» с кассеты.

7. Выполнять дамп модуля программы на машинном языке.

Предварительные требования для кода EZ

Прежде чем применять эту мониторную программу для ввода каких-либо программ на машинном языке, вы должны ассемблировать свою программу на языке ассемблера. Вам не нужно вычислять относительные или абсолютные переходы!

Ваш программный модуль не должен превышать 800 байтов или 200 команд.

Вы не должны загружать окончательный текст программы по адресам, меньшим 31499 (чтобы не стереть программу кода EZ).

Идейные основания программы кода EZ

Основная идея этой программы состоит в том, чтобы дать возможность вводить команды машинного языка в виде нумерованных строк, аналогично распечатке программы на языке «Бейсик».

Каждая строка «исходной программы» (так называются строки текста программы на машинном языке) имеет номер и содержит до 4 байтов текста.

Основное преимущество этого программного средства, таким образом, состоит в возможности «редактирования» любой строки. «Исходная программа» может также отдельно записываться на ленту, что позволяет запоминать работу на конкретных этапах.

Основная новизна этой программы — в возможности вставки относительных и абсолютных переходов без необходимости пересчета соответствующих чисел, любой переход можно сделать, просто сославшись на номер строки, на которую вы хотите перейти!

Это означает, что изменения можно вносить без труда даже в пределах диапазона относительного перехода.

Текст «исходной программы» передается в память по команде «DUMP». Получающийся в результате текст программы на машинном языке можно также записать в память.

Сводка команд кода EZ

Обратите внимание, что первый вопрос, задаваемый вам программой: «адрес загрузки».

Это адрес, по которому вы хотите разместить текст программы на машинном языке. Он не может быть меньше 31500.

Ввод строк

1. Для ввода строк «исходной программы»: (номер-строки) (пробел) (не более 4 шестнадцатеричных байтов) (ENTER) Например, 1 210040 приведет к вводу машинной команды LD HL, 4000H в строку номер 1.

2. Для редактирования строки: (номер строки) (пробел) (заново ввести новые БАЙТЫ) (ENTER) Например, 1 210140 приведет к замене строки номер 1 командой LD HL, 4001H

3. Для удаления строки команды: (НОМЕР-СТРОКИ) (ENTER) Например, 1 (ENTER) приведет к удалению строки номер 1.

4. Для задания относительного или абсолютного перехода (номер-строки) (пробел) (команда перехода) («L» в нижнем регистре») (номер-строки) (ENTER)

Например, 1 C312 представляет команду JP на строку 2. 2 1811 представляет команду JR на строку 1.

Команды

1. DUMP (ENTER)

Дамп исходной распечатки в память, начиная с заданного адреса загрузки. Это необходимо проделать перед выполнением программы на машинном языке. Сокращение: DU.

2. EXIT (ENTER)

Выход из кода EZ и повторный вход в систему «Бейсик». Сокращение: EX.

3. LIST (ENTER)

Распечатка первых 22 строк команд распечатки исходного текста. Нажмите любую клавишу, кроме «M» и «BREAK», чтобы продолжить распечатку. Сокращение: LI

LIST# (ENTER)

Распечатка 22 строк распечатки исходного текста начиная со строки N#, где номер — от 1 до 200 включительно. Без сокращения.

4. LOAD (ENTER) Загрузить модуль с распечаткой исходного текста с кассеты, заменив им существующий модуль. Сокращение: LD

5. MEM (ENTER) подсказка: адрес начала. Введите адрес памяти с которого вы хотите начать выдачу. Может быть от 0 до 32767 для «SPECTRUM» с объемом памяти 16K или от 0 до 65535 для «SPECTRUM» с объемом памяти 48K. Нажмите «M», чтобы выйти из режима просмотра памяти. Сокращение: ME.

6. NEW (ENTER) Удалить текущий модуль и заново стартовать код EZ. Эта команда полезна для того, чтобы начать ввод текста еще одного программного модуля. Сокращение: NE.

7. RUN (ENTER) Выполнить подвергнутый дампу программный модуль, начиная с адреса загрузки, заданного вами, когда стартовали программу кода EZ или при загрузке новой распечатки исходного текста. Сокращение: RU 8.

8. SAVE (ENTER)

Записать на кассету распечатку исходного текста или дамп текста машинной программы. Подсказка: ввести имя: Введите имя, которое вы хотите использовать, исходный текст или машинный язык: (S или M)

Введите S для записи распечатки исходного текста;

Введите M для записи текста на машинном языке;

Запустите ленту, потом нажмите любую клавишу, убедитесь, что кассета правильно заправлена, нажмите любую клавишу, когда кассета готова. Сокращение: SA

Замечания

1. Если вы не хотите, чтобы после выполнения программы возвращался результат в регистре BC, измените строку 3090 следующим образом:

3090 IF K$="RU" THEN LET L = USR R .

2. Для рестарта программы кода EZ : либо примените RUN, в результате чего все переменные будут инициализированы; либо примените GO TO 2020, в результате чего выдается подсказка «COMMAND OR LINE (###)».

3. Все вводимые числа, за исключением текста машинных команд должны быть в десятичном формате.

4. Чтобы вы могли вставлять дополнительные строки в текущую распечатку, полезно в распечатке пропускать номера строк, т.е. вместо того чтобы вводить строки команд с номерами 1, 2, 3, вводите 1, 5, 10 и т.д.

Это придаст большую гибкость вводимому модулю.

Упражнение на код EZ

Введите следующий текст программы:

```
210040      LD HL,4000H      :FILL SCREEN
110140      LD DE,4001H
01FF17      LD BC,6143
3EFF        LD A, 0FFH
77          LD (HL),A
EDB0        LDIR
3E7F        LOOP:LD A, 7FH   : TRAP BREAK KEY
DBFE        IN A,(0FEH)
E601        AND 1
20F8        JR NZ,LOOP
C9          RET
```

FILL SCREEN — заполнение экрана; TRAP BREAK KEY — прерывание по клавише. Чтобы ввести приведенный выше текст с помощью программы кода EZ — (RUN)

```
COMMAND OR LINE(###): 1  210040(ENTER)
COMMAND OR LINE(###): 5  110140(ENTER)
COMMAND OR LINE(###): 10 01FF17(ENTER)
COMMAND OR LINE(###): 15 3EFF(ENTER)
COMMAND OR LINE(###): 20 77(ENTER)
COMMAND OR LINE(###): 25 EDB0(ENTER)
COMMAND OR LINE(###): 30 3E7F(ENTER)
COMMAND OR LINE(###): 35 DBFE(ENTER)
COMMAND OR LINE(###): 40 E601(ENTER)
COMMAND OR LINE(###): 45 20130(ENTER)
```

(это будет 20, потом «L» в нижнем регистре, потом 30, т.е. JR NZ,LINE 30)

```
COMMAND OR LINE(###): 50 C9(ENTER)
COMMAND OR LINE(###): LIST(ENTER)
COMMAND OR LINE(###): DUMP(ENTER)
COMMAND OR LINE(###): MEM(ENTER)
STARTING ADDRESS: 31500(ENTER)

(THIS IS THE KEY TO EXIT THE MEMORY DISPLAY MODE)
COMMAND OR LINE(###): RUN(ENTER)
(BREAK)
```

RUN — выполнить; LOADING ADDRESS — адрес загрузки; COMMAND OR LINE — команда или строка; STARTING ADDRESS — адрес начала; THIS IS THE KEY TO EXIT THE MEMORY DISPLAY MODE — это — клавиша для выхода из режима просмотра памяти.

Обратите внимание, что после номеров строк должен идти пробел.

```
EZCODE
COPYRIGHT (C) 1982 BY WILLIAM TANG AND A. M. SULLIVAN
```

```
100 REM MACHINE
110 REM MACHINE CODE MONITOR
120 GO TO 9000
130 DEF FN D(S$) =(S$>"9"*(CODE S$-55)
   +(S$<="9")*(CODE S$-48)-(S$ > " ")*32
140 DEF FN 0(0$) = ((0$ = "CA")+(0$ = "DA")
   +(0$ = "EA")+(0$ = "FA")+(0$ = "C2")
   +(0$ = "D2")+(0$ = "E2")+(0$ = "F2")
   +(0$ = "C3"))-((0$ = "38")+(0$ = "30")
   +(0$ = "28")+(0$ = "20")+(0$ = "18")
   +(0$ = "10"))
1000 REM
1010 REM INV LINE PRINTING ROUTINE IRU
1020 CLS : PRINT AT ZE,25;INVERSE ON;FLASH ON;"LISTING IRU"
1030 LET F = ZE : PRINT AT ZE, ZE;
1040 FOR J = PL1 TO PL2
1050 IF C$(J,ON) = " " THEN GO TO 1110
1060 PRINT TAB TR- LEN STR$ J; J; TAB FR; " ";
1070 IF C$(J, TW, ON TO ON) ="1"
   THEN PRINT C$(J, ON)+" "+C$(J, TW)+C$(J, TR)
   : GO TO 1090)
1080 PRINT C$(J, ON); " "; C$(J, TW); " "
   ; C$(J, TR); " "; C$(J, TR)
1090 LET F =F+ON
1100 IF F = 22 THEN GO TO 1120
1110 NEXT J
1120 PRINT AT ZE, 25; " "
1130 RETURN
2000 REM INV MAIN ROUTINE IRU
2020 INPUT "COMMAND OR LINE(###) : "; A$
2030 IF A$( TO FR) =" "THEN GO TO MR
2040 IF A$(ON) > "9" THEN GO TO 3000
2050 LET K$ = "": FOR K = ON TO FR
2060 IF A$(K TO K) = " " THEN GO TO 2090
2070 LET K$ = K$+A$(K TO K)
2080 NEXT K
2090 IF K = 5 OR VAL K$ = ZE OR VAL K$ > IN
   THEN GO TO MR
2100 LET J = VAL K$ : LET N = J
   : REM LINE NUMBER MUST BE 3 BYTES
2110 LET A$ = A$(K+ON TO)
2120 LET K$ = ""
2130 FOR K = ON TO LEN A$
2140 IF A$(K TO K) <> " "
   THEN LET K$ =K$+A$(K TO K) 2150 NEXT K
2160 LET A$ = K$
2162 IF A$(ON) = "1" THEN GO TO MR
2170 CLS : FOR I = ON TO 7 STEP TW
2180 LET K = INT (I/TW+ON)
2190 LET C$(J,K) = A$(I TO I+ON)
```

```
2200 NEXT I
2210 IF C$(N,ON) = " " THEN GO TO 2250
2220 IF N < TP THEN LET TP = N
2230 IF N > BP THEN LET BP = N
2240 GO TO 2320
2250 IF N <> BP THEN GO TO 2280
2260 IF BP = ON OR C$(BP, ON) <> " "
   THEN GO TO 2320
2270 LET BP = BP-ON : GO TO 2260
2280 IF N <> TP THEN GO TO 2320
2290 IF C$(TP, ON) <> " " THEN GO TO 2320
2300 IF TP <> BP AND TP <> IN THEN LET TP = TP+ON
   : GO TO 2290
2310 LET TP = ON
2320 LET PP = N
2330 IF N < TP THEN LET PP = TP : GO TO 2380
2340 LET NUMLP = ZE
2350 IF PP = TP OR NUMLP = 11 THEN GO TO 2380
* 2360 IF C$(PP, ON) <> " "
   THEN LET NUMLP = NUMLP+ON
2370 LET PP = PP-ON : GO TO 2350
2380 LET PL1 = PP : LET PL2 = BP
2390 GO SUB 1000
   * REM PRINT A BLOCK OF LINES
2400 GO TO MR
3000 REM
3010 REM INV COMMANDS********** IRU
3020 LET K$ = A$( TO TW)
3030 IF K$ = "DU" THEN GO TO 5000
3040 IF K$ = "ex" THEN STOP
3050 IF K$ = "LI" THEN GO TO 4000
3060 IF K$ = "LO" THEN GO TO 7000
3070 IF K$ = "ме" THEN GO TO 6000
3080 IF K$ = "NE" THEN RUN
3090 IF K$ = "RU" THEN PRINT USR R
3100 IF K$ = "SA" THEN GO TO 8000
3110 GO TO MR
4000 REM
4000 REM INV LIST ROUTINE********** IRU
4020 LET PL1 = TP : LET PL2 = BP
4030 LET N1 = CODE A$(6 TO 6)
4040 IF LEN A$ > FR AND NL > 47 AND N1 < 58
   THEN LET PL1 = VAL A$(5 TO 8)
4050 GO SUB 1000
4060 GO TO MR
5000 REM
5010 REM INV DUMP ROUTINE********** IRU
5020 CLS : PRINT AT ZE, 25; INK ON; INVERSE ON
   ; FLASH ON; "DUMPING" : LET G = R
5030 PRINT AT ON, ZE;
```

```
5040 FOR J = TP TO BP
5050 IF C$(J, ON) = " " THEN GO TO 5470
5060 IF C$(J, TW, ON TO ON) <> "1" THEN GO TO 5380
5070 POKE G, ZE : POKE G+ON, ZE : POKE G+TW, ZE
   ; POKE G+TR, ZE
5080 LET JL = VAL (C$(J, TW, TW TO TW)+C$(J, TR))
5090 PRINT TAB TR- LEN STR$ J; INVERSE ON; J
   ; TAB FR; INVERSE ZE; " "
   ; C$(J, ON)+" "+C$(J, TW)+C$(J, TR)
   ; " = > ";
5100 IF JL < ZE OR JL > LN THEN GO TO 5460
5110 LET CJ = FN O(C$(J, ON))
5120 PRINT TAB 17- LEN STR$ JL; INVERSE ON; JL
   ; TAB 18; INVERSE ZE; " "; C$(JL, ON)
   ; " "; C$(JL, TW); " "; C$(JL, TR); " "
   ; C$(JL, FR);
5130 IF ABS CJ <> ON THEN GO TO 5460
5140 LET DD = (JL > J) - (JL < J)
5150 LET JA = G : LET DP = ZE
5170 LET CL = J+DD
5180 LET NL = ZE : IF C$(CL, ON) = " "
   THEN GO TO 5220
5190 IF C$(CL, TW, ON TO ON) <> "1"
   THEN LET NL = ON+(C$(CL, TW) <> " ")
   +(C$(CL, TR) <> " ")
   +(C$(CL, FR) <> " ")
   : GO TO 5220
5200 LET TJ = FN O(C$(CL, ON))
5210 LET N1 = (TJ = ON)*TR+(TJ = -ON)*TW
5220 IF CL = JL AND DD > ZE THEN GO TO 5270
5230 LET DP = DP+N1
5240 IF CL = JL THEN GO TO 5270
5250 LET CL = CL+DD
5260 GO TO 5180
5270 IF CJ = ON THEN LET JA = JA+DD*DP+(DD > ZE)*TR
   : GO TO 5310
5280 IF DD > ZE THEN LET DP = DP+2
5290 IF DP > 126 AND DD < ZE THEN GO TO 5460
5300 IF DP > 129 AND DD > ZE THEN GO TO 5460
5310 LET V = 16* FN D(C$(J, ON, ON TO ON))
   + FN D(C$(J, ON, TW TO TW))
5320 POKE G, V : LET G = G+ON
5330 IF CJ = ON THEN POKE G, JA- INT (JA/QK)*QK
   : LET G = G+ON : POKE G, INT (JA/QK)
   : LET G = G+ON : GO TO 5360
5340 IF DD < ZE THEN LET DP = -DP
5350 LET DP = DP-TW : POKE G, DP : LET G = G+ON
5360 PRINT "OK"
5370 GO TO 5470
5380 FOR I = ON TO 7 STEP TW
```

```
5390 LET K = INT (I/TW+ON)
5400 LET V = 16* FN D(C$(J, K, ON TO ON))
   + FN D(C$(J, K, TW TO TW))
5410 IF V < ZE THEN GO TO 5440
5420 POKE G, V
5430 LET G = G+ON
5440 NEXT I
5450 GO TO 5470
5460 PRINT "**"
5470 NEXT J
5480 PRINT AT ZE, 25; " "
   : GO TO MR
6000 REM
6010 REM INV MEMORY DISPLAY********* IBU
6020 INPUT "STARTING ADDRESS : "; DM
6030 CLS : PRINT AT ZE, ZE ;
6040 LET G = DM : LET F = ZE
6050 LET F = F+ON
   : PRINT TAB 5-LEN STR$ G; G; TAB 6;
6060 FOR I= ON TO FR
6070 LET V = PEEK G
6080 LET H = INT (V/16)
6090 LET L = V-16*H
6100 PRINT D$(H+ON) ; D$(L+ON ; " ";
6110 LET G = G+ON
6120 NEXT I
6130 PRINT " "
6140 IF F<> 22 THEN GO TO 6050
6050 LET K$ = INKEY$ : IF K$ ="" THEN GO TO 6150
6160 IF K$ <> "M" AND K$ <> "м" THEN LET F = ZE
   : POKE 23692, QK-ON : GO TO 6050
6200 POKE 23692, ON : PAUSE 20 : GO TO MR
7000 REM
7010 REM INV LOAD ****************** IRU
7020 CLS
7030 INPUT "LOAD ARRAY : PRESS ANY KEY WHEN READY. "
   ; K$
7040 PRINT AT ZE, 25; INVERSE ON; FLASH ON; "LOADING"
7050 LOAD "SOURCE" DATA C$()
7060 FOR I = ON TO IN
7070 LET TP = I
7080 IF C$(I, ON) <> " " THEN GO TO 7100
7090 NEXT I
7100 FOR I = IN TO ON STEP -1
7110 LET BP = I
7120 IF C$(I, ON) <> " " THEN GO TO 7140
7130 NEXT I
7140 PRINT AT ZE, 25; " "
7150 GO TO 9150
8000 REM
```

```
8010 REM INV SAVE****************** IRU
8020 INPUT "ENTER NAME : "; N$
8030 IF N$ = ""THEN GO TO 8020
8040 INPUT "SOURCE OR MACHIEN CODE : (S OR M)"
   ; K$
8050 IF K$ <> "S" AND K$ <> "M" THEN GO TO 8040
8060 IF K$ = "S" THEN SAVE N$ DATA C$() : GO TO MR
8070 INPUT "STARTING ADDRESS : "; SS
8080 INPUT "FINISHING ADDRESS : "; SF
8090 LET SB = SF-SS+ON
8100 SAVE N$ CODE SS, SB
8110 GO TO MR
9000 REM
9010 REM INITIALISATION
9020 LET ZE = PI - PI : LET ON = PI / PI
   : LET TW = ON+ON : LET TR = ON+TW
   : LET FR = TW+TW : LET QK = 256
   : LET MR = 2020 : LET LN = 200
9025 BORDER 7 : PAPER 7 : INK ON : INVERSE ZE
   : OVER ZE : FLASH ZE : BRIGHT ZE
   : BEEP .25, 24 : BEEP .25, 12
9030 DIM A$(15) : DIM O$(TW)
9040 LET TP = LN : LET BP = ON : REM LINE NUMBER BUFFE
9050 DIM C$(LN, FR, TW) : REM HOLDS CODE
9060 PRINT AT ZE, 20; INVERSE ON; FLASH ON
   ; "INITIALISING"
9070 FOR I = ON TO LN
9080 FOR J = ON TO FR
9090 LET C$(I, J) = " "
9100 NEXT J
9110 BEEP .01, 20
9120 NEXT I
9130 PRINT AT ZE, 20; " "
9140 LET D$ = "0123456789ABCDEF"
9150 CLS : PRINT "LOWEST ADDRESS : "; 31500
9160 INPUT "LOADING ADDRESS : "; R : PAUSE 20
9170 IF R < 31500 THEN GO TO 9160
9180 CLS : GO TO MR
```

COPYRIGHT — авторские права; MACHINE CODE MONITOR — монитор машинного языка; ROUTINE — программа; MAIN ROUTINE — основная программа; COMMAND OR LINE — команда или строка; NUMBER MUST BE 3 BYTES — номер строки должен составлять 3 байта; PRINT A BLOCK OF LINES — напечатать блок строк; COMMANDS — команды; MEMORY DISPLAY — распечатка памяти; STARTING ADDRESS — начальный адрес; LOAD — загрузка; LOAD ARRAY : PRESS ANY KEY WHEN READY — загрузка массива: нажмите любую клавишу, когда будете готовы; SAVE — запись; ENTER NAME — введите имя; SOURCE OR MACINE CODE — исходный текст или текст на машинном языке; STARTIN ADDRESS — начальный адрес; FINISHING ADDRESS — конечный адрес; INITIALISATION — инициализация; LINE NUMBER BUFFER — буфер номеров строк; LOWEST ADDRESS — наименьший адрес; LOADING ADDRESS — адрес загрузки.

Мониторная программа загрузки текста программы на машинном языке в шестнадцатеричном формате HEXLOAD

Эта написанная на языке «Бейсик» программа может быть монитором сама по себе, поскольку она может записывать данные в шестнадцатеричном формате в память, распечатывать содержимое памяти, перемещать содержимое памяти, записывать содержимое памяти на кассету и загружать его с кассеты.

С другой стороны, мы можем применять программу HEXLOAD в качестве полусвязывающего загрузчика (SIME-LINKING LOADER) для текста программы, созданного программой кода EZ. Так получается потому, что программа кода EZ может применяться только для ввода небольших модулей, не превышающих 800 байтов или 200 команд.

Так что для больших программ мы применяем программу кода EZ для разработки модулей и записываем каждый модуль в виде текста на машинном языке на кассету.

Затем мы используем программу HEXLOAD, гораздо меньшую по объему программу на языку «Бейсик», чтобы загрузить эти модули и связать их, переместив в отведенные им ячейки памяти.

Идейные основания программы HEXLOAD

Идеи, лежащие в основе программы HEXLOAD, чрезвычайно просты.

Мониторная программа на самом деле устанавливает RAMTOP системы «Бейсик» равным 26999.

Это означает, что вы можете вводить свою программу на машинном языке в любое место между ячейками 27000 и 32578 для «SPECTRUM» с 16K оперативной памяти или с 27000 по 65343 для «SPECTRUM» с 48K.

HEXLOAD — достаточно простая мониторная программа для работ с текстом на машинном языке. Она представляет такие базисные функции управления:

WRITE — запись в память в шестнадцатеричном формате;
SAVE — запись из памяти на кассету;
LOAD — запись с кассеты в память;
LIST — распечатка содержимого памяти начиная с начального адреса;
MOVE — перемещение содержимого памяти из одной группы ячеек в другую.

Сводка команд программы

1. WRITE Запись текста программы в шестнадцатеричном формате в память.

а) В ответ на подсказку ввести начальный адрес памяти, с которого вы хотите вести запись, в десятичном формате.

Адрес должен лежать в диапазоне:

27000 — 32578 для памяти объемом 16K;
27000 — 65346 для памяти объемом 48K.

Например: запись по адресу: 27000(ENTER)

б) Введите текст программы в шестнадцатеричном формате.

в) Нажмите клавишу «м», чтобы вернуться к основному меню.

2. SAVE

Запись содержимого памяти на кассету. Процедура:

а) Вводимое значение начального адреса памяти, с которого начинается запись, может быть любым из следующего диапазона:

0 — 32767 для памяти объемом 16K
0 — 65535 для памяти объемом 48K

б) Введите количество записываемых байтов.

в) Введите имя записываемого модуля.

г) Нажмите любую клавишу, когда кассета будет готова.

д) Возможность проверки модуля, записанного на кассету. Неплохо сделать проверку, чтобы удостовериться, что модуль не испортился во время процедуры записи.

3. LOAD

Загрузка модуля на машинном языке с кассеты. Процедура: а) введите адрес памяти, с которого модуль начинает загружаться. Адрес должен лежать в том же диапазоне, что и для команд записи.

б) введите имя, использованное при записи модуля. Если вы не уверены в имени, просто нажмите клавишу (ENTER).

4. LIST

Выдача содержимого памяти, начиная с некоторого адреса. Процедура:

а) Введите адрес начала распечатки. Это может быть любой адрес как в приведенной выше команде SAVE.

б) Нажмите любую клавишу для продолжения выдачи.

в) Нажмите клавишу «М», чтобы вернуться к основному меню.

5. MOVE

Переместить содержимое памяти с начального по конечный адрес по новому адресу в памяти. Процедура:

а) Ввести начальный адрес перемещаемой порции, любой адрес, как в диапазоне для команды SAVE

б) Ввести конечный адрес перемещаемой порции, любой адрес, как в диапазоне для команды SAVE.

в) Введите адрес памяти, куда перемещается информация, диапазон адресов, как в команде WRITE.

г) Вы можете даже с помощью этой команды копировать из ПЗУ в память с произвольным доступом.

Например:

Переместить, начиная с: 0(ENTER)
Переместить, кончая: 1000(ENTER)
Переместить по адресу: 32000(ENTER)

Такая последовательность действий приведет к перемещению содержимого ПЗУ с 0 по 1000 адрес в память с произвольным доступом по адресу 32000. Замечания: любая попытка ввода в приведенных выше командах нарушающая допустимый диапазон адресов, приведет к повторению подсказки для ввода.

Упражнение

Попробуйте с помощью этого монитора ввести модуль, разработанный нами с помощью программы кода EZ.

```
HEXLOAD
COPYIRIHT (C) 1982 BY WILLIAM TANG AND DAVID WEBB

100 REM
110 REM MONITOR PROGRAM
120 CLEAR 26999 : LET ZE = PI - PI
    : LET ON = PI / PI : LET TW = ON+ON
    : LET QK = 256 : LET LM = 27000
    : LET MR = 140 : LET WL = 340
130 GO SUB 2000
```

```
140 CLS: PRINT "START OF,MACHINE CODE AREA = " ; LM
150 PRINT "MENU" : PRINT : PRINT " WRITE MACHINE CODE.....1"
160 PRINT : PRINT " SAVE MACHINE CODE.....2"
170 PRINT : PRINT" LOAD MACHINE CODE.....3"
180 PRINT : PRINT " LIST MACHINE CODE.....4"
190 PRINT : PRINT " MOVE MACHINE CODE.....5"
200 PRINT : PRINT "PLEASE PRESS APPROPRIATE KEY."
210 LET G$ = INKEY$
220 IF G$ = " " OR G$ = "M" THEN STOP
230 IF G$ = "" OR G$ < "1" OR G$ > "5" THEN GO TO 210
240 CLS : PRINT "START OF MACHINE CODE AREA = " ; LM
250 GO TO 300* VAL G$
300 REM INV WRITE********** IRU
310 INPUT "WRITE TO ADDRESS :"; D
320 IF D > MM OR D < LM THEN GO TO 310
330 PRINT : PRINT "WRITE ADDRESS : "; D
    : PRINT "TO RETURN TO MENU ENTER ""M"""
340 LET A$ = ""
350 IF A$ = "" THEN INPUT "ENTER HEX. CODE :" ; A$
360 IF A$(ON) =" " OR A$(ON) = "M" THEN GO TO MR
370 IF LEN A$/TW <> INT (LEN A$/TW) THEN PRINT "INCORRECT
ENTRY ";
    : GO TO WL
380 LET C = ZE
390 FOR F = 16 TO ON STEP -15
400 LET A = CODE A$((F = 16)+TW*(F = ON))
410 IF A < 48 OR A > 102 OR (A > 57 AND A < 65)
    OR (A > 70 AND A < 97)
    THEN PRINT "INCORRECT ENTRY ";:GO TO WL
420 LET c = C+F*((A < 58)*(a-48)
    +(A > 64 AND A < 71)*(A-55)+(A < 96)*(A-87))
430 NEXT F : POKE D; C : LET D = D+ON
440 PRINT A$(TO TW); " ";
450 LET A$ =A$(3 TO )
460 IF D = UDG THEN PRINT
    "WARNING :YOU ARE NOW IN THE USER GRAPHICS AREA!": GO TO WL
470 IF D = UDR-20 THEN PRINT
    "WARNING : YOU ARE NOW IN ROUTINES MEMORY AREA!": GO TO WL
480 GO TO WL+ON
600 REM INV SAVE********** IRU
610 INPUT "SAVE M.C. FROM ADDRESS :"; A
620 INPUT "NUMBER OF BYTES TO BE SAVED :"; N
630 INPUT "NAME OF THE ROUTINE :"; A$
640 SAVE A$ CODE A, N
650 PRINT "DO YOU WISH TO VERITY?"
660 INPUT V$
670 IF V$ <> "Y" THEN GO TO MR
680 PRINT "REWIND TAPE AND PRESS ""PLAY""."
690 VERIFY A$ CODE A, N
700 PRINT "O.K." : PAUSE 50
```

```
710 GO TO MR
900 REM INV LOAD********** IRU
910 INPUT "LOAD M.C. TO ADDRESS STARTING : " ; A
920 IF A > MM OR A < LM THEN GO TO 910
930 INPUT "PROGRAM NAME :"; A$
940 PRINT "PRESS ""PLAY"" ON TAPE."
950 LOAD A$ CODE A : GO TO MR
1200 REM INV LIST********** IRU
1210 LET A$ = "0123456789ABCDEF"
1220 INPUT "LIST ADDRESS :"; D
1230 PRINT "PRESS ""M"" TO RETURN TO MENU."
1240 LET A = INT (PEEK D/16): LET B = PEEK D-16* INT(PEEK D/16)
1250 PRINT D; TAB 7; A$(A+ON); A$(B+ON)
1260 LET D = D+ON
1270 IF INKEY$ = " " OR INKEY$ = "M" THEN GO TO M
1280 GO TO 1240
1500 REM INV MOVE********** IRU
1510 INPUT "MOVE FROM MEMORY : "; FM
1520 INPUT "MOVE UNTIL MEMORY : "; UM
1530 INPUT "MOVE TO MEMORY : "; TM
1540 IF TM > FM THEN GO TO 1610
1550 LET MO = TM
1560 FOR I = FM TO UM
1570 POKE MP, PEEK I
1580 LET MP = MP+ON
1590 NEXT I
1600 GO TO MR
1610 LET MP = UM+TM-FM
1620 FOR I = UM to FM STEP -ON
1630 POKE MP, PEEK I
1640 LET MP = MP-ON
1650 NEXT I
1660 GO TO MR
2000 LET RT = PEEK 23732+QK* PEEK 23733
2010 IF RT = 65535 THEN LET MM = 65347
   : LET UDG = 65367
2020 IF RT = 32767 THEN LET MM = 32579
   : LET UDG = 32599
2030 LET N1 = INT (UDG/QK)
2040 POKE 23675, UDG-N1*QK : POKE 23676, N1
2050 RETURN
```

COPYRIGHT — авторские права; MONITOR PROGRAM — мониторная программа; START OF, MACHINE CODE AREA — начало области текста в машинных командах; MENU — меню; WRITE MACHINE CODE — записать текст на машинном языке; SAVE MACHINE CODE — запомнить текст на машинном языке; LOAD MACHINE CODE — загрузить текст на машинном языке; LIST MACHINE CODE — распечатать текст на машинном языке; MOVE MACHINE CODE — переместить текст на машинном языке; PLEASE PRESS APPROPRIATE KEY — пожалуйста, нажмите соответствующую клавишу; START OF MACHINE CODE AREA — начало области текста в машинных командах; WRITE TO ADDRESS — записать по адресу; TO RETURN TO MENU ENTER — чтобы вернуться к меню,

введите; ENTER HEX. CODE — введите в шестнадцатеричных кодах; INCORRECT ENTRY — неверный ввод; WARNING: YOU ARE NOW IN USER GRAPHICS AREA предупреждение: вы сейчас в графической области пользователя; WARNING: YOU ARE NOW IN THE ROUTINES MEMORY AREA — предупреждение: вы сейчас в области памяти программ 1; SAVE M. C. FROM ADDRESS — запомнить текст на машинном языке; NUMBER OF BYTES TO BE SAVED — количество запоминаемых байтов; NAME OF THE ROUTINE — название программы; DO YOU WISH TO VERIFY — хотите проверить? REWIND TAPE AND PRESS "PLAY" — перемотайте пленку и нажмите «пуск»; LOAD M.C. TO ADDRESS STARTING — загрузить текст на машинном языке по адресу; PROGRAM NAME — название программы; PRESS "PLAY" ON TAPE — нажмите «пуск» для пленки; LIST ADDRESS — адрес для распечатки; PRESS "M" TO RETURN TO MENU — нажмите «М», чтобы вернуться к меню; MOVE FROM MEMORY — переместить, начиная с ячейки памяти; MOVE UNTIL MEMORY — переместить до ячейки памяти; MOVE TO MEMORY — переместить по адресу.

Приложение А

Таблица клавиш «СПЕКТРУМ»

INPUT VALUE IN A FOR OFF H	D4	D3	D2	D1	D0
0FE H	V	C	X	Z	CAP SHIFT
0FD H	G	F	D	S	A
0FB H	7	R	E	W	Q
0F7 H	5	4	3	2	1
0EF H	6	7	8	9	0
0DF H	Y	U	I	O	P
0BF H	H	J	K	L	ENTER
07F H	B	N	M	SYM SHIFT	BREAK SPACE
X:	16	8	4	2	1

INPUT VALUE IN A OFF — входное значение в А для; CAP — прописные буквы; SHIFT — смена регистра; SYM — сокращение расшифровать не удалось; ENTER — ввод; BREAK — прерывание; SPACE — пробел.

Примечание: чтобы выполнить прерывание по клавише:

1. Загрузите в регистр А входное значение из соответствующего ряда.

LD A 07E : нижний ряд

2. Примите информацию с входного порта 0FEH.

IN A (0FEH)

3. Проверьте, что для нужной клавиши DX имеет низкое значение.

AND 1 : прерывание по клавише BREAK/SPACE

4. Если ноль, то клавиша нажата.

JR Z, клавиша нажата: в нормальном состоянии значение всегда высокое

Приложение B

MEMORI IN HEX	ATTRIBUTE IN HEX	LINE		ME-MORI IN HEX	ATTRIBU TE IN HEX
4000	5800	0		401F	581F
4020	5820	1		403F	583F
4040	5840	2		405F	585F
4060	5860	3		407F	587F
4080	5880	4		409F	589F
40A0	58A0	5		40BF	58BF
40C0	58C0	6		40DF	58DF
40E0	58E0	7		40FF	58FF
4800	5900	8		481F	591F
4820	5920	9		483F	593F
4840	5940	10		485F	595F
4860	5960	11		487F	597F
4880	5980	12		489F	599F
48A0	59A0	13		48BF	59BF
48C0	59C0	14		48DF	59DF
48E0	59E0	15		48FF	59FF
5000	5A00	16		501F	5A1F
5020	5A20	17		503F	5A3F
5040	5A40	18		505F	5A5F
5060	5A60	19		507F	5A7F
5080	5A80	20		509F	5A9F
50A0	5AA0	21		50BF	5ABF
50C0	5AC0	22		50DF	5ADF
50E0	5AE0	23		50FF	5AFF

MEMORY IN HEX — память в шестнадцатеричном формате; ATTRIBUTE IN HEX — атрибут в шестнадцатеричном формате; LINE — строка.

Таблица набора литер «СПЕКТРУМ»

HEX LOB	HOB	0	1	2	3	4	5	6	7
	BITS	000	001	010	011	100	101	110	111
0	00 00	NU	INK CTRL	S PC	0		P	,	
1	00 01	NU	RAPER CTRI	!	1	A	Q	,	
2	00 10	NU	FLASH CTRI	«	2	B	R	,	
3	00 11	NU	BRIGHT CTRI	#	3	C	S	,	
4	01 00	NU	INVERSE CTRI	S	4	D	T	,	
5	01 01	NU	OVER CTRI	%	5	E	U	,	
6	01 10	PRINT	AT CTRI	$	6	F	V	,	
7	01 11	EDIT	TAB CTRI	»	7	G	W	,	
8	10 00	CURSOR LEFT	NU	(	8	H	X	,	
9	10 01	CURSOR RIGHT	NU	)	9	I	Y	,	
A	10 10	CURSOR DOWN	NU	.	:	J	Z	,	
B	10 11	CRSOR UP	NU	+	;	K		,	
C	11 00	DELETE	NU	,	<	L		,	
D	11 01	ENTER	NU	-	=	M		,	
E	11 10	NUMBER	NU	.	>	N		,	
F	11 11	NU	NU	/	?	.P	–	,	

NB: NU = NOT USED.

NEX — шестнадцатеричное представление; LOB — младший байт; HOB — старший байт; INK CTRI — управление чернилами; SPACE — пробел; PAPER CTRI — управление

бумагой; FLASH CTRI — управление режимом мигания; BRIGHT CTRI — управление яркостью; INVERSE CTRI — управление инверсией яркости; OVER CTRI — управление превышением; PRINT — печать; AT CTRI — управление; EDIT — редактирование; TAB CTRI — управление табуляцией; CURSOR LEFT — курсор влево; CURSOR RIGHT — курсор вправо; CURSOR DOWN — курсор вниз; CURSOR UP — курсор вверх; DELETE — удаление; ENTER — ввод; NUMBER — число; NON PRINTABLE — непечатные; PRINTABLE — печатные; NB: — замечание; NU — не используется.

Приложение D

Таблицы преобразования десятичных чисел в шестнадцатеричные

HEX	0	1	2	3	4	5	6	7	8	9	A
0	0	1	2	3	4	5	6	7	8	9	10
1	16	17	18	19	19	20	21	22	23	24	25 2

HEX — шестнадцатеричное.

Мы можем показать применение этой таблицы на примере. Давайте найдем шестнадцатеричный эквивалент десятичного числа 6200. Нам нужно определить 16-битовое двоичное число, т. е.,

```
0001BBBB    BBBBBBBB
HOB         LOB
```

HOB — старший байт; LOB — младший байт

1. Из самой левой колонки таблицы под заголовком XX00 мы находим, что 6200 находится между 4096 и 8192. Так что мы выбираем меньшее значение 4096 и из значения ряда мы берем

4 самых старших бита старшего байта равные 1, т. е. 01.

```
BBBBBBBB    BBBBBBBB
HOB         LOB
```

HOB — старший байт; LOB — младший байт;

2. Второй шаг состоит в том, что мы определяем следующие по старшинству 4 бита старшего байта. Мы находим разность между 6200 и 4096, равную 2104. Поскольку разность все еще превышает 255, мы обращаемся ко второй слева колонке таблицы под заголовком 00XX и выясняем, что 2104 находится между 2048 и 2304. Вновь мы выбираем меньшее значение 2048 и по значению ряда получаем, что следующее по старшинству 4 бита старшего байта равны 8, т.е. 1000.

```
00011000    BBBBBBBB
HOB         LOB
```

HOB — старший байт; LOB — младший байт.

3. Третий шаг состоит в определении младшего байта числа. Мы обнаруживаем, что разность между 2104 и 2048 равна 56 лежит на пересечении ряда 3 и колонки 8. Так что мы принимаем младший байт равным 38H.

```
00011000    00111000
HOB         LOB
```

HOB — старший байт; LOB — младший байт.

Итак, шестнадцатеричное значение числа 6200 равно 1838H.

Таблица преобразования десятичных чисел в форме дополнения до 2 в шестнадцатеричный формат

HEX	8	9	A	B	C	D	E	F
0	-128	-112	-96	-80	-64	-48	-32	-16
1	-127	-111	-95	-79	-63	-47	-31	-15
2	-126	-110	-94	-78	-62	-46	-30	-14
3	-125	-109	-93	-77	-61	-45	-29	-13
4	-124	-108	-92	-76	-60	-44	-28	-12
5	-123	-107	-91	-75	-59	-43	-27	-11
6	-122	-106	-90	-74	-58	-42	-26	-10
7	-121	-105	-89	-73	-57	-41	-25	-9
8	-120	-104	-88	-72	-56	-40	-24	-8
9	-119	-103	-87	-71	-55	-39	-23	-7
A	-118	-102	-86	-70	-54	-38	-22	-6
B	-117	-101	-85	-69	-53	-37	-21	-5
C	-116	-100	-84	-68	-52	-36	-20	-4
D	-115	-99	-83	-67	-51	-35	-19	-3
E	-114	-98	-82	-66	-50	-34	-18	-2
F	-113	-97	-81	-65	-49	-33	-17	-1

HEX — шестнадцатеричное

Приложение F

Шестнадцатеричная таблица сложения

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13		

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Приложение G

Сводная таблица изменения флагов

INSTRUCTION	C	Z	P/V	S	N	H	COMMENTS
ADC NL, SS	#	#	V	#	0	X	16-BIT ADD WITH CARRY
ADX S, ADD S	#	#	V	#	0	#	8-BIT ADD OR ADD WITH CARRY
ADD DD, SS	#	-	-	-	0	X	16-BIT ADD
AND S	0	#	P	#	0	1	LOGICAL OPERATIONS
BIT B,,S	-	#	X	X	0	1	STATE OF BIT B OF LOICATION S IS COPIED IINTO THE Z FLAG
CCF	#	-	-	-	0	X	COMPLEMENT CARRY
CPD:CPDR:CPI:CPIR	-	#	#	X	1	X	BLOK SEARCH INSTRUCTION Z=1;IF A=(HL),, ELSE Z=0 P/V=1; IF BC#G,,OTHERVISE P/V=0
CP S	#	#	V	#	1	#	COMPARE ACCUMULATOR
CPL	-	-	-	-	1	1	COMPLEMENT ACCUMULATOR
DAA	#	#	P	#	-	#	DECIMAL ABJUST ACCUMULATOR
DEC S	-	#	V	#	1	#	8-BIT DECREMENT

INSTRUCTION	C	Z	P/V	S	N	H	COMMENTS
IN R,,(C)	-	#	P	#	0	0	INPUT REGISTER INDIRECT
INC S	-	#	V	#	0	#	8-BIT INCREMENT
IND; INI	-	#	X	X	1	X	BLOK INPUT Z=0 IF B#0 ELSE Z=1
INDR; INIR	-	1	X	X	1	X	BLOK INPUT Z=0 IF B#0 ELSE Z=1
LDA,,I;LDA,,R	-	#	I FF	#	0	0	CONTENT OF INTERRUPT ENADLE FLIP-FLOP IS COPIED INTO THE P/V FLAG
LDD; LDI	-	X	#	X	0	0	BLOCK TRANSFER INSTRUCTIONS
LDDR; LDIR	-	X	0	X	0	0	P/V=1 IF BC#0,,OTHERWISE P/V=0
NEG	#	#	V	#	1	#	NEGATE ACCUMULATOR
OR S	0	#	P	#	0	0	LOGICAL OR ACCUMULATOR
OTDR;OTIR	-	1	X	X	1	X	BLOK OUTPUT;Z=0 IF B#0 OTHERWISE Z=1
OUTD;OUTI	-	#	X	X	1	X	BLOK OUTPUT;Z=0 IF B#0 OTHERWISE Z=1
RLA;RLCA;RRA;RRCA	#	-	-	-	0	0	ROTATE ACCUMULATOR
RLD; RRD	-	#	P	#	0	/	ROTATE DIGIT LEFT AND RIGHT
RLS;RLC S;RR S;	#	#	P	#	0	0	ROTATE AND SHIFT
RRC S;SLA S;SRA S							LOCATION S
SRL S							
SBC HL; SS	#	#	V	#	1	X	16-BIT SUBTRACT WITH CARRY
SCF	1	-	-	-	0	0	SET CARRY
SBC S; SUB S			V	1			8-BIT SUBTRACT WITH CARRY
XOR X	0		P		0	0	EXCLUSIVE OR ACCUMULATOR

INSTRUCTION — команда; COMMENTS — комментарии; 16-BIT ADD WITH CARRY — 16-битовое сложение с переносом; 8-BIT ADD OR ADD WITH CARRY — 8-битовое

сложение или сложение с переносом; 16-BIT ADD — 16-битовое сложение; LOGICAL OPERATION — логические операции; STATE OF BIT B OF LOCATION S IS COPED INTO THE Z FLAG — состояние бита в ячейки S копируется в флаг Z; COMPLEMENT CARR — дополнение переноса; BLOCK SEARCH INSTRUCTION — команда блочного поиска; IF — если; ELSE — иначе; OTHERWISE — в противном случае; COMPARE ACCUMULATOR — сравнение накапливающего регистра COMPLEMENT ACCUMULATOR — дополнение накапливающего регистра; DECIMAL ADJUST ACCUMULATOR — десятичная настройка накапливающего регистра; 8-BIT DECREMENT — 8-битовое уменьшение; INPUT REGISTER INDIRECT — косвенный ввод регистра; 8-BIT INCREMENT — 8-битовое увеличение; BLOCK INPUT — ввод блока; CONTENT OF INTERRUPT ENABLE FLIP-FLOP IS COPIED INTO THE P/V FLAG — содержимое триггера прерывания копируется в P/V флаг; BLOCK TRANSFER INSTRUCTIONS — команды передачи блока; NEGATE ACCUMULATOR — отрицание для накапливающего регистра; LOGICAL OR ACCUMULATOR — логическое или для накапливающего регистра; BLOCK OUTPUT — вывод блока; ROTATE ACCUMULATOR — циклический сдвиг накапливающего регистра; ROTATE DIGIT LEFT ANT RIGHT — циклический сдвиг разряда влево или вправо; ROTATE AND SHIFT LOCATION — циклический сдвиг и сдвиг ячейки; 16-BIT SUBTRACT WITH CARRY — 16-битовое вычитание с переносом; SET CARRY — установка переноса; 8-BIT SUBTRACT WITH CARRY — 8-битовое вычитание с переносом; EXCLUSIVE OR ACCUMULATOR — исключающее ИЛИ для накапливающего регистра.

C флаг переноса. C=1, если операция привела к переносу из самого значащего бита операнда или результата.

Z флаг нуля. Z=1,если результат операции нулевой.

S флаг знака. S=1, если самый значащий бит результата равен единице, т.е. число отрицательное.

P/V флаг четности или переполнения. Четность (P) и переполнение (0) относятся к одному и тому же флагу. Для логических операций этот флаг задает четность результата, а для арифметических — переполнение.

Если в P/V хранится четность: P/V=1, если результат операции четный, P/V=0, если результат нечетный.

Если P/V содержит переполнение: P/V=1, если в результате операции получилось переполнение.

H флаг половинного переноса. H=1, если при операции сложения или вычитания произошел перенос или заем в четвертом бите накапливающего регистра

N флаг сложения (вычитания). N=1, если предыдущей операцией было вычитание. Флаг N и H используются в совокупности с командой десятичной настройки (DAA) для исправления результата в правильной упакованной двоично-кодированный десятичный формат после сложения или вычитания с применением операндов в упакованном двоично-кодированном десятичном формате.

\# флаг устанавливается в соответствии с результатом операции.

– флаг не изменяется в результате операции.

0 флаг сбрасывается (=0) операцией.

1 флаг устанавливается (=1) операцией.

X флаг результата неизвестен.

V действие на флаг P/V соответствует результату переполнения при операции.

P действие на флаг P/V соответствует четности результата операции.

R любой из регистров ЦП: A, B, C, D, E, N, L.

S любая 8-битовая ячейка для всех режимов адресации, допустимых для конкретной команды.

Команды ЦП Z80 в порядке возрастания кодов операций

HEXADECIMAL	MNEMONIC
00	NOP
01XXXX	LD BC,NN
02	LD(BC),A
03	INC BC
04	INC B
05	DEC B
06XX	LD B,N
07	RLCA
08	EX AF,AF"
09	ADD HL,BC
0A	LD A,(BC)
0B	DEC BC
0C	INC C
0D	DEC C
0EXX	LD C,N
0F	RRCA
10XX	DJNZ DIS
11XXXX	LD DE,NN
12	LD (DE),A
13	INC DE
14	INC D
15	DEC D
16XX	LD D,N
17	RLA
18XX	JR DIS
19	ADD HL,DE
1A	LD A,(DE)
1B	DEC DE
1C	INC E
1D	DEC E
1EXX	LD E,N
1F	RRA
20XX	JR NZ,DIS
21XXXX	LD HL,NN
22XXXX	LD (NN),HL
23	INC HL
24	INC H
25	DEC H
26XX	LD H,N
27	DAA
28XX	JR Z,DIS
29	ADD HL,HL
2AXXXX	LD HL(NN)
2B	DEC HL
2C	INC L
2D	DEC L
2EXX	LD L,N
2F	CPL
30XX	JR NC,DIS
31XXXX	LD SP,NN
32XXXX	LD (NN),A
33	INC SP
34	INC (HL)
35	DEC (HL)
3620XX	LD (HL),N
37	SCF
38XX	JR C,DIS
39	ADD HL,SP
3AXXXX	LD A,(NN)
3B	DEC SP
3C	INC A
3D	DEC A
3EXXXX	LD A
3F	CCF
40	LD B,B
41	LD B,C
42	LD B,D
43	LD B,E
44	LD B,H
45	LD B,L
46	LD B,(HL)
47	LD B,A
48	LD C,B
49	LD C,C
4A	LD C,D
4B	LD C,E
4C	LD C,H
4D	LD C,L
4E	LD C,(HL)
4F	LD C,A
50	LD D,B
51	LD D,C
52	LD D,D
53	LD D,E
54	LD D,H
55	LD D,L
56	LD D,(HL)
57	LD D,A
58	LD E,B
59	LD E,C

5A	LD E,D
5B	LD E,E
5C	LD E,H
5D	LD E,L
5E	LD E,(HL)
5F	LD E,A
60	LD H,B
61	LD H,C
62	LD H,D
63	LD H,E
64	LD H,H
65	LD H,L
66	LD H,(HL)
67	LD H,A
68	LD L,B
69	LD L,C
6A	LD L,D
6B	LD L,E
6C	LD L,H
6D	LD L,L
6E	LD L,(HL)
6F	LD L,A
70	LD (HL),B
71	LD (HL),C
72	LD (HL),D
73	LD (HL),E
74	LD (HL),H
75	LD (HL),L
76	HALT
77	LD (HL),A
78	LD A,B
79	LD A,C
7A	LD A,D
7B	LD A,E
7C	LD A,H
7D	LD A,L
7E	LD A,(HL)
7F	LD A,A
80	ADD A,B
81	ADD A,C
82	ADD A,D
83	ADD A,E
84	ADD A,H
85	ADD A,L
86	ADD A,(HL)
87	ADD A,A
88	ADC A,B
89	ADC A,C
8A	ADC A,D
8B	ADC A,E
8C	ADC A,H
8D	ADC A,L
8E	ADC A,(HL)
8F	ADC A,A
90	SUB B
91	SUB C
92	SUB D
93	SUB E
94	SUB H
95	SUB L
96	SUB (HL)
97	SUB A
98	SBC A,B
99	SBC A.C
9A	SBC A,D
9B	SBC A,E
9C	SBC A,H
9D	SBC A,L
9E	SBC A,(HL)
9F	SBC A,A
A0	AND B
A1	AND C
A2	AND D
A3	AND E
A4	AND H
A5	AND L
A6	AND (HL)
A7	AND A
A8	XOR B
A9	XOR C
AA	XOR D
AB	XOR E
AC	XOR H
AD	XOR L
AE	XOR (HL)
AF	XOR A
B0	OR B
B1	OR C
B2	OR D
B3	OR E
B4	OR H
B5	OR L
B6	OR (HL)
B7	OR A
B8	CP B
B9	CP C
BA	CP D
BB	CP E
BC	CP H
BD	CP L

BE	CP (HL)
BF	CP A
C0	RET NZ
C1	POP BC
C2XXXX	JP NZ,NM
C3XXXX	JP NM
C4XXXX	CALL NZ,NM
C5	PUSH BCHL
C6XX	ADD A,N
C7	RET 0
C8	RET Z
C9	RET
CAXXXX	JP Z,NM
CB00	RLC 8
CB01	RLC C
CB02	RLC D
CB03	RLC E
CB04	RLC H
CB05	RLC L
CB06	RLC (HL)
CB07	RLC A
CB08	RRC B
CB09	RRC C
CB0A	RRC D
CB0B	RRC E
CB0C	RRC H
CB0D	RRC L
CB0E	RRC (HL)
CB0F	RRC A
CB10	RL B
CB11	RL C
CB12	RL D
CB13	RL E
CB14	RL H
CB15	RL L
CB16	RL (HL)
CB17	RL A
CB18	RR B
CB19	RR C
CB1A	RR D
CB1B	RR E
CB1C	RR H
CB1D	RR L
CB1E	RR (HL)
CB1F	RR A
CB20	SLA B
CB21	SLA C
CB22	SLA D
CB23	SLA E
CB24	SLA H
CB25	SLA L
CB26	SLA (HL)
CB27	SLA A
CB28	SRA B
CB29	SRA C
CB2A	SRA D
CB2B	SRA E
CB2C	SRA H
CB2D	SRA L
CB2E	SRA (HL)
CB2F	SRA A
CB38	SRL B
CB39	SRL C
CB3A	SRL D
CB3B	SRL E
CB3C	SRL H
CB3D	SRL L
CB3E	SRL (HL)
CB3F	SRL A
CB40	BIT 0,B
CB41	BIT 0,C
CB42	BIT 0,D
CB43	BIT 0,E
CB44	BIT 0,H
CB45	BIT 0,L
CB46	BIT 0,(HL)
CB47	BIT 0,A
CB48	BIT 1,B
CB49	BIT 1,C
CB4A	BIT 1,D
CB4B	BIT 1,E
CB4C	BIT 1,H
CB4D	BIT 1,L
CB4E	BIT 1,(HL)
CB4F	BIT 1,A
CB50	BIT 2,B
CB51	BIT 2,C
CB52	BIT 2,D
CB53	BIT 2,E
CB54	BIT 2,H
CB55	BIT 2,L
CB56	BIT 2,(HL)
CB57	BIT 2,A
CB58	BIT 3,B
CB59	BIT 3,C
CB5A	BIT 3,D
CB5B	BIT 3,E
CB5C	BIT 3,H
CB5D	BIT 3,L
CB5E	BIT 3,(HL)

CB5F	BIT 3,A
CB60	BIT 4,B
CB61	BIT 5,C
CB62	BIT 4,D
CB63	BIT 4,E
CB64	BIT 4,H
CB65	BIT 4,L
CB66	BIT 4,(HL)
CB67	BIT 4,A
CB68	BIT 5,B
CB69	BIT 5,C
CB6A	BIT 5,D
CB6B	BIT 5,E
CB6C	BIT 5,H
CB6D	BIT 5,L
CB6E	BIT 5,(HL)
CB6F	BIT 5,A
CB70	BIT 6,B
CB71	BIT 6,C
CB72	BIT 6,D
CB73	BIT 6,E
CB74	BIT 6,H
CB75	BIT 6,L
CB76	BIT 6,(HL)
CB77	BIT 6,A
CB78	BIT 7,B
CB79	BIT 7,C
CB7A	BIT 7,D
CB7B	BIT 7,E
CB7C	BIT 7,H
CB7D	BIT 7,L
CB7E	BIT 7,(HL)
CB7F	BIT 7,A
CB80	RES 0,B
CB81	RES 0,C
CB82	RES 0,D
CB83	RES 0,E
CB84	RES 0,H
CB85	RES 0,L
CB86	RES 0,(HL)
CB87	RES 0,A
CB88	RES 1,B
CB89	RES 1,C
CB8A	RES 1,D
CB8B	RES 1,E
CB8C	RES 1,H
CB8D	RES 1,L
CB8E	RES 1,(HL)
CB8F	RES 1,A
CB90	RES 2,B
CB91	RES 2,C
CB92	RES 2,D
CB93	RES 2,E
CB94	RES 2,H
CB95	RES 2,L
CB96	RES 2,(HL)
CB97	RES 2,A
CB98	RES 3,B
CB99	RES 3,C
CB9A	RES 3,D
CB9B	RES 3,E
CB9C	RES 3,H
CB9D	RES 3,L
CB9E	RES 3,(HL)
CB9F	RES 3,A
CBA0	RES 4,B
CBA1	RES 4,C
CBA2	RES 4,D
CBA3	RES E,E
CBA4	RES E,H
CBA5	RES 4,L
CBA6	RES 4,(HL)
CBA7	RES 4,A
CBA8	RES 5,B
CBA9	RES 5,C
CBAA	RES 5,D
CBAB	RES 5,E
CBAC	RES 5,H
CBAD	RES 5,L
CBAE	RES 5,(HL)
CBAF	RES 5,A
CBB0	RES 6,B
CBB1	RES 6,C
CBB2	RES 6,D
CBB3	RES 6,E
CBB4	RES 6,H
CBB5	RES 6,L
CBB6	RES 6,(HL)
CBB7	RES 6,A
CBB8	RES 7,B
CBB9	RES 7,C
CBBA	RES 7,D
CBBB	RES 7,E
CBBC	RES 7,H
CBBD	RES 7,L
CBBE	RES 7,(HL)
CBBF	RES 7,A
CBC0	SET 0,B
CBC1	SET 0,C
CBC2	SET 0,D

CBC3	SET 0,E
CBC4	SET 0,H
CBC5	SET 0,L
CBC6	SET 0,(HL)
CBC7	SET 0,A
CBC8	SET 1,B
CBC9	SET 1,C
CBCA	SET 1,D
CBCB	SET 1,E
CBCC	SET 1,H
CBCD	SET 1,L
CBCE	SET 1,(HL)
CBCF	SET 1,A
CBD0	SET 2,B
CBD1	SET 2,C
CBD2	SET 2,D
CBD3	SET 2,E
CBD4	SET 2,H
CBD5	SET 2,L
CBD6	SET 2,(HL)
CBD7	SET 2,A
CBD8	SET 3,B
CBD9	SET 3,C
CBDA	SET 3,D
CBDB	SET 3,E
CBDC	SET 3,H
CBDD	SET 3,L
CBDE	SET 3,(HL)
CBDF	SET 3,A
CBE0	SET 4,B
CBE1	SET 4,C
CBE2	SET 4,D
CBE3	SET 4,E
CBE4	SET 4,H
CBE5	SET 4,L
CBE6	SET 4,(HL)
CBE7	SET 4,A
CBE8	SET 5,B
CBE9	SET 5,C
CBEA	SET 5,D
CBEB	SET 5,E
CBEC	SET 5,H
CBED	SET 5,L
CBEE	SET 5,(HL)
CBEF	SET 5,A
CBF0	SET 6,B
CBF1	SET 6,C
CBF2	SET 6,D
CBF3	SET 6,E
CBF4	SET 6,H
CBF5	SET 6,L
CBF6	SET 6,(HL)
CBF7	SET 6,A
CBF8	SET 7,B
CBF9	SET 7,C
CBFA	SET 7,D
CBFB	SET 7,E
CBFC	SET 7,H
CBFD	SET 7,L
CBFE	SET 7,(HL)
CBFF	SET 7,A
CCXXXX	CALL Z,NN
CDXXXX	CALL NN
CEXX	ADC A,N
CF	RST 8
D0	RET NC
D1	POP DE
D2XXXX	JP NC,NN
D3XX	OUT (N),A
D4XXXX	CALL NC,NN
D5	PUSH DE
D6XX	SUB N
D7	RST 10H
D8	RET C
D9	EXX
DAXXXX	JP C,NN
DBXX	IN A,(N)
DCXXXX	CALL C,NN
DD09	ADD IX,BC
DD19	ADD IX,DE
DD21XXXX	LD IX,NN
DD22XXXX	LD (NN),IX
DD23	INC IXC),L
DD29	ADD IX,IX
DD2AXXXX	LD IX,(NN)
DD2B	DEC IX
DD34XX	INC (IX+Д)
DD35XX	DEC (IX+Д)
DD36XX20	LD (IX+Д),N
DD39	ADD IX,SP
DD46XX	LD B,(IX+Д)
DD4EXX	LD C,(IX+Д)
DD56XX	LD D,(IX+Д)
DD5EXX	LD E,(IX+Д)
DD66XX	LD H,(IX+Д)
DD6EXX	LD L,(IX+Д)
DD70XX	LD (IX+Д),B
DD71XX	LD (IX+Д),C
DD72XX	LD (IX+Д),D
DD73XX	LD (IX+Д),E

DD74XX	LD (IX+Д),H
DD75XX	LD (IX+Д),L
DD77XX	LD (IX+Д),A
DD7EXX	LD A,(IX+Д)
DD86XX	ADD A,(IX+Д)
DD8EXX	ADC A,(IX+Д)
DD96XX	SUB (IX+Д)
DD9EXX	SBC A,(IX+Д)
DDA6XX	AND (IX+Д)
DDAEXX	XOR (IX+Д)
DDB6XX	OR (IX+Д)
DDBEXX	CP (IX+Д)
DDCBXX06	RLC (IX+Д)
DDCBXX0E	RRC (IX+Д)
DDCBXX16	RL (IX+Д)
DDCBXX1E	RR (IX+Д)
DDCBXX26	SLA (IX+Д)
DDCBXX2E	SRA (IX;D)
DDCBXX3E	SRL (IX+Д)
DDCBXX46	BIT0,(IX+Д)
DDCBXX4E	BIT1,(IX+Д)
DDCBXX56	BIT2,(IX+Д)
DDCBXX5E	BIT3,(IX+Д)
DDCBXX66	BIT4,(IX+Д)
DDCBXX6E	BIT5,(IX+Д)
DDCBXX76	BIT6,(IX+Д)
DDCBXX7E	BIT7,(IX+Д)
DDCBXX86	RES0,(IX+Д)
DDCBXX8E	RES1,(IX+Д)
DDCBXX96	RES2,(IX+Д)
DDCBXX9E	RES3,(IX+Д)
DDCBXXA6	RES4,(IX+Д)
DDCBXXAE	RES5,(IX+Д)
DDCBXXB6	RES6,(IX+Д)
DDCBXXBE	RES7,(IX+Д)
DDCBXXC6	SET0,(IX+Д)
DDCBXXCE	SET1,(IX+Д)
DDCBXXD6	SET2,(IX+Д)
DDCBXXDE	SET3,(IX+Д)
DDCBXXE6	SET4,(IX+Д)
DDCBXXEE	SET5,(IX+Д)
DDCBXXF6	SET6,(IX+Д)
DDCBXXFE	SET7,(IX+Д)
DDE1	POP IX
DDE3	EX (SP),IX
DDE5	PUSH IX
DDE9	JP (IX)
DDF9	LD SP,IX
DEXX	SBC A,N
DF	RST 18H

E0	RET PO
E1	POP HL
E2XXXX	JP PO,NN
E3	EX (SP),HL
E4XXXX	CALL PO,NN
E5	PUSH HL
E6XX	AND N
E7	RST 20H
E8	RET PE
E9	JP (HL)
EAXXXX	JE PE NN
EB	EX DE,HL
ECXXXX	CALL PE,NN!
ED40	IN B,(C)
ED41	OUT (C),B
ED42	SBC HL,BC
ED43XXXX	LD (NN),IX
ED44	NEG
ED45	RETN
ED46	IM 0
ED47	LD I,A
ED48	INC C,(C)
ED49	OUT (C),C
ED4A	ADC HL,BC
ED4BXXXX	LD BC,(NN)
ED4D	RET I
ED50	IN D,(C)
ED51	OUT (C),D
ED52	SBC HL,DE
ED53XXXX	LD (NN),DE
ED56	IN I
ED57	LD A,I
ED58	IN E,(C)
ED59	OUT (C),E
ED5A	ADC HL,DE
ED5BXXXX	LD DE,(NN)
ED5E	IN 2
ED60	IN H,(C)
ED61	OUT (C),H
ED62	SBC HL,HL
ED67	RRD
ED68	IN L,(C)
ED69	OUT (C),L
ED6A	ADC HL,HL
ED6F	RLD
ED72	SBC HL,SP
ED73XXXX	LD (NN),SP
ED78	IN A,(C)
ED79	OUT (C),A
ED7A	ADC HL,SP

ED7BXXXX	LD SP,(NN)
EDA0	LDI
EDA1	CPI
EDA2	INI
EDA3	OUTI
EDA8	LDD
EDA9	CP0
EDAA	IND
EDAB	OUTD
ED80	LDIR
ED81	CPIR
ED82	INIR
ED83	OTIR
ED88	LDDR
ED89	CPDR
ED8A	INDR
ED8B	OTDR
EEXX	XOR N
EF	RST 28H
F0	RET P
F1	POP AF
F2XXXX	JR P,NN
F3	D1
F4XXXX	CALL P,NN
F5	RUSH AF
F620XX	OR N
F7	RST 30H
F8	RET N
F9	LD SP,HL
FAXXXX	JP N,NN
FB	E1
FCXXXX	CALL M,NN
FE20XX	CP N
FF	RST 38H
FD09	ADD IY,BC
FD19	ADD IY,DC
ED21XXXX	LD IY,NN
FD22XXXX	LD (NN),IY
FD23	INC IY
FD29	ADD IY,IY
FD2AXXXX	LD IY,(NN)
FD2B	DEC IY
FD34XX	INC (IY+Д)
FD35XX	DEC (IY+Д)
FD36XX20	LD (IY+Д),N
FD39	ADD IY,CP
FD46XX	LD B,(IY+Д)
FD4EXX	LD C,(IY+Д)
FD56XX	LD D,(IY+Д)
FD5EXX	LD E,(IY+Д)
FD66XX	LD H,(IY+Д)
FD6EXX	LD L,(IY+Д)
FD70XX	LD (IY+Д),B
FD71XX	LD (IY+Д),C
FD72XX	LD (IY+Д),D
FD73XX	LD (IY+Д),E
FD74XX	LD (IY+Д),H
FD75XX	LD (IY+Д),L
FD77XX	LD (IY+Д),A
FD7EXX	LD A,(IY+Д)
FD86XX	ADD A,(IY+Д)
FD8EXX	ADC A,(IY+Д)
FD96XX	SUB (IY+Д)
FD9EXX	SBC A,(IY+Д)
FDA6XX	AND (IY+Д)
FDAEXX	XOR (IY+Д)
FDB6XX	OR (IY+Д)
FDBEXX	CP (IY+Д)
FDE1	POP IY
FDE3	EX (SP),IY
FDE5	RUSH IY
FDE9	JP (IY)
FDF9	LD SP,IY
FDCBXX06	RLC (IY+Д)
FDCBXX0E	RRC (IY+Д)
FDCBXX16	RL (IY+Д)
FDCBXX1E	RR (IY+Д)
FDCBXX26	SLA (IY+Д)
FDCBXX2E	SRA (IY+Д)
FDCBXX3E	SRL (IY+Д)
FDCBXX46	BIT 0,(IY+Д)
FDCBXX4E	BIT 1,(IY+Д)
FDCBXX56	BIT 2,(IY+Д)
FDCBXX5E	BIT 3,(IY+Д)
FDCBXX66	BIT 4,(IY+Д)
FDCBXX6E	BIT 5,(IY+Д)
FDCBXX76	BIT 6,(IY+Д)
FDCBXX7E	BIT 7,(IY+Д)
FDCBXX86	RES 0,(IY+Д)
FDCBXX8E	RES 1,(IY+Д)
FDCBXX96	RES 2,(IY+Д)
FDCBXX9E	RES 3,(IY+Д)
FDCBXXA6	RES 4,(IY+Д)
FDCBXXAE	RES 5,(IY+Д)
FDCBXXB6	RES 6,(IY+Д)
FDCBXXBE	RES 7,(IY+Д)
FDCBXXC6	SET 0,(IY+Д)
FDCBXXCE	SET 1,(IY+Д)
FDCBXXD6	SET 2,(IY+Д)
FDCBXXDE	SET 3,(IY+Д)

FDCBXXE6	SET 4,(IY+Д)
FDCBXXEE	SET 5,(IY+Д)
FDCBXXF6	SET 6,(IY+Д)
FDCBXXFE	SET 7,(IY+Д)

Команды ЦП Z80 в порядке возрастания мнемонических обозначений.

MNEMONIC	HEXADECIMAL
ADC A,(HL)	8E
ADC A,(IX+Д)	DD8EXX
ADC A,(IY+Д)	FD8EXX
ADC A,A	8F
ADC A,B	88
ADC A,C	89
ADC A,D	8A
ADC A,N	CEXX
ADC A,E	8B
ADC A,H	8C
ADC A,L	8D
ADC HL,BC	ED4A
ADC HL,DE	ED5A
ADC HL,HL	ED6A
ADD HL,SP	ED7A
ADD A,(HL)	86
ADD A,(IX+Д)	DD86XX
ADD A,(IY+Д)	FD86XX
ADD A,A	87
ADD A,B	80
ADD A,C	81
ADD A,D	82
ADD A,N	C6XX
ADD A,E	83
ADD A,H	84
ADD A,L	85
ADD HL,BC	09
ADD HL,DE	19
ADD HL,HL	29
ADD HL,SP	39
ADD IX,BC	DD09
ADD IX,DE	DD19
ADD IX,IX	DD29
ADD IX,SP	DD39
ADD IY,BC	FD09
ADD IY,DE	FD19
ADD IY,IY	FD29
ADD IY,SP	FD39
AND (HL)	A6
AND (IX+Д)	DDA6XX
AND (IY+Д)	FDA6XX
AND A	A7
AND B	A0
AND C	A1
AND D	A2
AND N	E6XX
AND E	A3
AND H	A4
AND L	A5
BIT 0,(HL)	CB46
BIT0,(IX+Д)	DDCBXX46
BIT0,(IY+Д)	FDCBXX46
BIT 0,A	CB47
BIT 0,B	CB40
BIT 0,C	CB41
BIT 0,D	CB42
BIT 0,E	CB43
BIT 0,H	CB44
BIT 0,L	CB45
BIT 1,(HL)	CB4E
BIT 1,(IX+Д)	DDCBXX4E
BIT 1,(IY+Д)	FDCBXX4E
BIT 1,A	CB4F
BIT 1,B	CB48
BIT 1,C	CB49
BIT 1,D	CB4A
BIT 1,E	CB4B
BIT 1,H	CB4C
BIT 1,L	CB4D
BIT 2,(HL)	CB56
BIT 2,(IX+Д)	DDCBXX56
BIT 2,(IY+Д)	FDCBXX56
BIT 2,A	CB57
BIT 2,B	CB50
BIT 2,C	CB51
BIT 2,D	CB52
BIT 2,E	CB53
BIT 2,H	CB54
BIT 2,L	CB55
BIT 3,(HL)	CB5E

BIT 3,(IX+Д)	DDCBXX5E
BIT 3,(IY+Д)	FDCBXX5E
BIT 3,A	CB5F
BIT 3,B	CB58
BIT 3,C	CB59
BIT 3,D	CB5A
BIT 3,E	CB5B
BIT 3,H	CB5C
BIT 3,L	CB5D
BIT 4,(HL)	CB66
BIT 4,(IX+Д)	DDCBXX66
BIT 4,(IY+Д)	FDCBXX66
BIT 4,A	CB67
BIT 4,B	CB60
BIT 4,C	CB61
BIT 4,D	CB62
BIT 4,E	CB63
BIT 4,H	CB64
BIT 4,L	CB65
BIT 5,(HL)	CB6E
BIT 5,(IX+Д)	DDCBXX6E
BIT 5,(IY+Д)	FDCBXX6E
BIT 5,A	CB6F
BIT 5,B	CB68
BIT 5,C	CB69
BIT 5,D	CB6A
BIT 5,E	CB6B
BIT 5,H	CB6C
BIT 5,L	CB6D
BIT 6,(HL)	CB76
BIT6,(IX+Д)	DDCBXX7
BIT6,(IY+Д)	FDCBXX7
BIT 6,A	CB77
BIT 6,B	CB70
BIT 6,C	CB71
BIT 6,D	CB72
BIT 6,E	CB73
BIT 6,H	CB74
BIT 6,L	CB75
BIT 7,(HL)	CB7E
BIT7,(IX+Д)	DDCBXX7
BIT7,(IY+Д)	FDCBXX7
BIT 7,A	CB7F
BIT 7,B	CB78
BIT 7,C	CB79
BIT 7,D	CB7A
BIT 7,E	CB7B
BIT 7,H	CB7C
BIT 7,L	CB7D
CALL ADDR	CDXXXX
CALL C,ADDR	DCXXXX
CALL M,ADDR	FCXXXX
CALL NC,ADDR	D4XXXX
CALL NZ,ADDR	C4XXXX
CALL P,ADDR	F4XXXX
CALL PE,ADDR	ECXXXX
CALL PO,ADDR	E4XXXX
CALL Z,ADDR	CCXXXX
CPF	3F
CP (HL)	BE
CP (IX+Д)	DDBEXX
CP (IY+Д)	FDBEXX
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP N	FEXX
CP E	BB
CP H	BC
CP L	BD
CPD	EDA9
CPDR	EDB9
CPI	EDA1
CPIR	EDB1
CPL	2F
DAA	27
DEC (HL)	35
DEC (IX+Д)	DD35XX
DEC (IY+Д)	FD35XX
DEC A	3D
DEC B	05
DEC BC	0B
DEC C	0D
DEC D	15
DEC DE	1B
DEC E	1D
DEC H	25
DEC HL	2B
DEC IX	DD2B
DEC IY	FD2B
DEC L	2D
DEC SP	3B
DI	F3
DJNZ,Д	10XX
EI	F8
EX (SP),HL	E3
EX (SP),IX	DDE3
EX (SP),IY	FDE3
EX AF,AF'	08
EX DE,HL	EB

EXX	D9
HALT	76
IM 0	ED46
IM 1	ED56
IM 2	ED5E
IN A,(C)	ED78
IN A,PORT	D8XX
IN B,(C)	ED40
IN C,(C)	ED48
IN D,(C)	ED50
IN E,(C)	ED58
IN H,(C)	ED60
IN L,(C)	ED68
INC (HL)	34
INC (IX+Д)	DD34XX
INC (IY+Д)	FD34XX
INC A	3C
INC B	04
INC BC	03
INC C	0C
INC D	14
INC DE	13
INC E	1C
INC H	24
INC HL	23
INC IX	DD23
INC IY	FD23
INC L	2C
INC SP	33
IND	EDAA
INCR	EDBA
INI	EDA2
INIR	EDB2
JP (HL)	E9
JP (IX)	DDE9
JP (IY)	FDE9
JP ADDR	C3XXXX
JP C,ADDR	DAXXXX
JP M,ADDR	FAXXXX
JP NC,ADDR	D2XXXX
JP NZ,ADDR	C2XXXX
JP P,ADDR	F2XXXX
JP PE,ADDR	EAXXXX
JP PO,ADDR	E2XXXX
JP Z,ADDR	CAXXXX
JR C,Д	38XX
JR Д	18XX
JR NC,Д	30XX
JR NZ,Д	20XX
JR Z,Д	28XX
LD (ADDR),A	32XXXX
LD(ADDR),BC	ED43XXXX
LD(ADDR),DE	ED53XXXX
LD(ADDR),HL	ED63XXXX
LD(ADDR),HL	22XXXX
LD(ADDR),IX	DD22XXXX
LD(ADDR),IY	FD22XXXX
LD(ADDR),SP	ED73XXXX
LD (BC),A	02
LD (DE),A	12
LD (HL),A	77
LD (HL),B	70
LD (HL),C	71
LD (HL),D	72
LD (HL),N	36XX
LD (HL),E	73
LD (HL),H	74
LD (HL),L	75
LD (IX+Д),A	DD77XX
LD (IX+Д),B	DD70XX
LD (IX+Д),C	DD71XX
LD (IX+Д),D	DD72XX
LD (IX+Д),N	DD36XXXX
LD (IX+Д),E	DD73XX
LD (IX+Д),H	DD74XX
LD (IX+Д),L	DD75XX
LD (IY+Д),A	FD77XX
LD (IY+Д),B	FD70XX
LD (IY+Д),C	FD71XX
LD (IY+Д),D	FD72XX
LD (IY+Д),N	FD36XXXX
LD (IY+Д),E	FD73XX
LD (IY+Д),H	FD74XX
LD (IY+Д),L	FD75XX
LD A,(ADDR)	3AXXXX
LD A,(BC)	0A
LD A,(DE)	1A
LD A,(HL)	7E
LD A,(IX+Д)	DD7EXX
LD A,(IY+Д)	FD7EXX
LD A,A	7F
LD A,B	78
LD A,C	79
LD A,D	7A
LD A,E	7B
LD A,H	7C
LD A,I	ED57
LD A,L	7D
LD A,N	3EXX
LD A,R	ED5F

LD B,(HL)	46
LD B,(IX+Д)	DD46XX
LD B,(IY+Д)	FD46XX
LD B,A	47
LD B,B	40
LD B,C	41
LD B,D	42
LD B,N	06XX
LD B,E	43
LD B,H	44
LD B,L	45
LD BC,(ADDR)	ED4BXXXX
LD BC,NN	01XXXX
LD C,(HL)	4E
LD C,(IX+Д)	DD4EXX
LD C,(IY+Д)	FD4EXX
LD C,A	4F
LD C,B	48
LD C,C	49
LD C,D	4A
LD C,N	0EXX
LD C,E	4B
LD C,H	4C
LD C,L	4D
LD D,(HL)	56
LD D,(IX+Д)	DD56XX
LD D,(IY+Д)	FD56XX
LD D,A	57
LD D,B	50
LD D,C	51
LD D,D	52
LD D,N	16XX
LD D,E	53
LD D,H	54
LD D,L	55
LD DE,(ADDR)	ED5BXXXX
LD DE,NN	11XXXX
LD E,(HL)	5E
LD E,(IX+Д)	DD5EXX
LD E,(IY+Д)	FD5EXX
LD E,A	5F
LD E,B	58
LD E,C	59
LD E,D	5A
LD E,N	1EXX
LD E,E	5B
LD E,H	5C
LD E,L	5D
LD H,(HL)	66
LD H,(IX+Д)	DD66XX
LD H,(IY+Д)	FD66XX
LD H,A	67
LD H,B	60
LD H,C	61
LD H,D	62
LD H,N	26XX
LD H,E	63
LD H,H	64
LD H,L	65
LDHL,(ADDR)	ED68XXXX
LDHL,(ADDR)	2AXXXX
LD HL,NN	21XXXX
LD I,A	ED47
LDIX,(ADDR)	DD2AXXXX
LDIX,NN	DD21XXXX
LDIY,(ADDR)	FD2AXXXX
LD IY,NN	FD21XXXX
LD L,A	6F
LD L,B	68
LD L,C	69
LD L,D	6A
LD L,N	2EXX
LD L,E	6B
LD L,(HL)	6E
LDL,(IX+Д)	DD6EXX
LDL,(IY+Д)	FD6EXX
LD L,H	6C
LD L,L	6D
LD R,A	ED4F
LDSP,(ADDR)	ED7BXXXX
LD SP,NN	31XXXX
LD SP,HL	F9
LD SP,IX	DDF9
LD SP IY	FDF9
LDD	EDA8
LDDR	ED88
LDI	EDA0
LDIR	EDB0
NEG	ED44
NOP	00
OR (HL)	B6
OR (IX+Д)	DDB6XX
OR (IY+Д)	FDB6XX
OR A	B7
OR B	B0
OR C	B1
OR D	B2
OR N	F6XX
OR E	B3
OR H	B4

OR L	B5	RES 2,C	CB91
OTDR	EDBB	RES 2,D	CB92
OTIR	EDB3	RES 2,E	CB93
OUT (C),A	ED79	RES 2,H	CB94
OUT (C),B	ED41	RES 2,L	CB95
OUT (C),C	ED49	RES 3,(HL)	CB9E
OUT (C),D	ED51	RES3,(IX+Д)	DDCBXX9E
OUT (C),E	ED59	RES3,(IY+Д)	FDCBXX9E
OUT (C),H	ED61	RES 3,A	CB9F
OUT (C),L	ED69	RES 3,B	CB98
OUT PORT,A	D3PORT	RES 3,C	CB99
OUTD	EDAB	RES 3,D	CB9A
OUTI	EDA3	RES 3,E	CB9B
POP AF	F1	RES 3,H	CB9C
POP BC	C1	RES 3,L	CB9D
POP DE	D1	RES 4,(HL)	CBA6
POP HL	E1	RES4,(IX+Д)	DDCBXXA6
POP IX	DDE1	RES4,(IY+Д)	FDCBXXA6
POP IY	FDE1	RES 4,A	CBA7
PUSN AF	F5	RES 4,B	CBA0
PUSH BC	C5	RES 4,C	CBA1
PUSH DE	D5	RES 4,D	CBA2
PUSH HL	E5	RES 4,E	CBA3
PUSH IX	DDE5	RES 4,H	CBA4
PUSH IY	FDE5	RES 4,L	CBA5
RES 0,(HL)	CB86	RES 5,(HL)	CBAE
RES0,(IX+Д)	DDCBXX86	RES5,(IY+Д)	DDCBXXAE
RES0,(IY+Д)	FDCBXX86	RES5,(IY+Д)	FDCBXXAE
RES 0,A	CB87	RES 5,A	CBAF
RES 0,B	CB80	RES 5,B	CBA8
RES 0,C	CB81	RES 5,C	CBA9
RES 0,D	CB82	RES 5,D	CBAA
RES 0,E	CB83	RES 5,F	CBAB
RES 0,H	CB84	RES 5,H	CBAC
RES 0,L	CB85	RES 5,L	CBAD
RES 1,(HL)	CB8E	RES 6,(HL)	CBB6
RES1,(IX+Д)	DDCBXX8E	RES6,(IX+Д)	DDCBXXB6
RES1,(IY+Д)	FDCBXX8E	RES6,(IY+Д)	FDCBXXB6
RES 1,A	CB8F	RES 6,A	CBB7
RES 1,B	CB88	RES 6,B	CBB0
RES 1,C	CB89	RES 6,C	CBB1
RES 1,D	CB8A	RES 6,D	CBB2
RES 1,E	CB8B	RES 6,E	CBB3
RES 1,H	CB8C	RES 6,H	CBB4
RES 1,L	CB8D	RES 6,L	CBB5
RES 2,(HL)	CB96	RES 7,(HL)	CBBE
RES2,(IX+Д)	DDCBXX96	RES7,(IX+Д)	DDCBXXBE
RES2,(IY+Д)	FDCBXX96	RES7,(IY+Д)	FDCBXXBE
RES 2,A	CB97	RES 7,A	CBBF
RES 2,B	CB90	RES 7,B	CBB8

RES 7,C	CBB9
RES 7,D	CBBA
RES 7,E	CBBB
RES 7,H	CBBC
RES 7,L	CBBD
RET	C9
RET C	D8
RET M	F8
RET NC	D0
RET NZ	C0
RET P	F0
RET PE	E8
RET PO	E0
RET Z	C8
RETI	ED4D
RETN	ED45
RL (HL)	CB16
RL (IX+Д)	DDCBXX16
RL (IY+Д)	FDCBXX16
RL A	CB17
RL B	CB10
RL C	CB11
RL D	CB12
RL E	CB13
RL H	CB14
RL L	CB15
RLA	17
RLC (HL)	CB06
RLC (IX+Д)	DDCBXX06
RLC (IY+Д)	FDCBXX06
RLC A	CB07
RLC B	CB00
RLC C	CB01
RLC D	CB02
RLC E	CB03
RLC H	CB04
RLC L	CB05
RLCA	07
RLD	ED6F
RR (HL)	CB1E
RR (IX+Д)	DDCBXX1E
RR (IY+Д)	FDCBXX1E
RR A	CB1F
RR B	CB18
RR C	CB19
RR D	CB1A
RR E	CB1B
RR H	CB1C
RR L	CB1D
RRA	1F

RRC (HL)	CB0E
RRC (IX+Д)	DDCBXX0E
RRC (IY+Д)	FDCBXX0E
RRC A	CB0F
RRC B	CB08
RRC C	CB09
RRC D	CB0A
RRC E	CB0B
RRC H	CB0C
RRC L	CB0D
RRCA	0F
RRD	ED67
RST 00	C7
RST 08	CF
RST 10	D7
RST 18	DF
RST 20	E7
RST 28	EF
RST 30	F7
RST 38	FF
SBC A,(HL)	9E
SBC A,(IX+Д)	DD9EXX
SBC A,(IY+Д)	FD9EXX
SBC A,A	9FXX
SBC A,B	98
SBC A,C	99
SBC A,D	9A
SBC A,N	DEXX
SBC A,E	9B
SBC A,H	9C
SBC A,L	9D
SBC HL,BC	ED42
SBC HL,DE	ED52
SBC HL,HL	ED62
SBC HL,SP	ED72
SCF	37
SET 0,(HL)	CBC6
SET0,(IX+Д)	DDCBXXC6
SET0,(IY+Д)	FDCBXXC6
SET 0,A	CBC7
SET 0,B	CBC0
SET 0,C	CBC1
SET 0,D	CBC2
SET 0,E	CBC3
SET 0,H	CBC4
SET 0,L	CBC5
SET 1,(HL)	CBCE
SET1,(IX+Д)	DDCBXXCE
SET1,(IY+Д)	FDCBXXCE
SET 1,A	CBCF

SET 1,B	CBC8
SET 1,C	CBC9
SET 1,D	CBCA
SET 1,E	CBCB
SET 1,H	CBCC
SET 1,L	CBCD
SET 2,(HL)	CBD6
SET2,(IX+Д)	DDCBXXD6
SET2,(IY+Д)	FDCBXXD6
SET 2,A	CBD7
SET 2,B	CBD0
SET 2,C	CBD1
SET 2,D	CBD2
SET 2,E	CBD3
SET 2,H	CBD4
SET 2,L	CBD5
SET 3,(HL)	CBDE
SET3,(IX+Д)	DDCBXXCE
SET3,(IY+Д)	FDCBXXCE
SET 3,A	CBDF
SET 3,B	CBD8
SET 3,C	CBD9
SET 3,D	CBDA
SET 3,E	CBDB
SET 3,H	CBDC
SET 3,L	CBDD
SET 4,(HL)	CBE6
SET4,(IX+Д)	DDCBXXE6
SET4,(IY+Д)	FDCBXXE6
SET 4,A	CBE7
SET 4,B	CBE0
SET 4,C	CBE1
SET 4,D	CBE2
SET 4,E	CBE3
SET 4,H	CBE4
SET 4,L	CBE5
SET 5,(HL)	CBEE
SET5,(IX+Д)	DDCBXXEE
SET5,(IY+Д)	FDCBXXEE
SET 5,A	CBEF
SET 5,B	CBE8
SET 5,C	CBE9
SET 5,D	CBEA
SET 5,E	CBEB
SET 5,H	CBEC
SET 5,L	CBED
SET 6,(HL)	CBF6
SET6,(IX+Д)	DDCBXXF6
SET6,(IY+Д)	FDCBXXF6
SET 6,A	CBF7
SET 6,B	CBF0
SET 6,C	CBF1
SET 6,D	CBF2
SET 6,E	CBF3
SET 6,H	CBF4
SET 6,L	CBF5
SET 7,(HL)	CBFE
SET7,(IX+Д)	DDCBXXFE
SET7,(IY+Д)	FDCBXXFE
SET 7,A	CBFF
SET 7,B	CBF8
SET 7,C	CBF9
SET 7,D	CBFA
SET 7,E	CBFB
SET 7,H	CBFC
SET 7,L	CBFD
SLA (HL)	CB26
SLA (IX+Д)	DDCBXX26
SLA (IY+Д)	FDCBXX26
SLA A	CB27
SLA B	CB20
SLA C	CB21
SLA D	CB22
SLA E	CB23
SLA H	CB24
SLA L	CB25
SRA (HL)	CB2E
SRA (IX+Д)	DDCBXX2E
SRA (IY+Д)	FDCBXX2E
SRA A	CB2F
SRA B	CB28
SRA C	CB29
SRA D	CB2A
SRA E	CB2B
SRA H	CB2C
SRA L	CB2D
SRL (HL)	CB3E
SRL (IX+Д)	DDCBXX3E
SRL (IY+Д)	FBCBXX3E
SRL A	CB3F
SRL B	CB38
SRL C	CB39
SRL D	CB3A
SRL E	CB3B
SRL H	CB3C
SRL L	CB3D
SUB (HL)	96
SUB (IX+Д)	DD96XX
SUB (IY+Д)	FD96XX
SUB A	97

SUB B	90
SUB C	91
SUB D	92
SUB N	D6XX
SUB E	93
SUB H	94
SUB L	95
XOR (HL)	AE
XOR (IX+Д)	DDAEXX
XOR (IY+Д)	FDAEXX
XOR A	AF
XOR B	A8
XOR C	A9
XOR D	AA
XOR N	EEXX
XOR E	AB
XSOR H	AC
XOR L	AD

MNEMONIC — мнемоника;
HEAXDECIMAL — шестнадцатеричный код.
Д - приращение

для заметок